



AFRL-RY-WP-TR-2009-1033



REINCARNATION OF STREAMING APPLICATIONS

Saman Amarsinghe, Robert Miller, and Michael Ernst

Massachusetts Institute of Technology

OCTOBER 2009

Final Report

Approved for public release; distribution unlimited.

See additional restrictions described on inside pages

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Defense Advanced Research Projects Agency (DARPA) and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RY-WP-TR-2009-1033 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

*//Signature//

ALFRED J. SCARPELLI
Project Engineer
Advanced Sensor Components Branch
Aerospace Components & Subsystems
Technology Division

//Signature//

BRADLEY J. PAUL, Chief
Chief, Advanced Sensor Components Branch
Aerospace Components & Subsystems
Technology Division
Sensors Directorate

//Signature//

TODD A. KASTLE
Chief, Aerospace Components & Subsystems
Technology Division
Sensors Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

*Disseminated copies will show “//Signature//” stamped or typed above the signature blocks.

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YY) October 2009		2. REPORT TYPE Final		3. DATES COVERED (From - To) 27 September 2007 – 31 December 2008	
4. TITLE AND SUBTITLE REINCARNATION OF STREAMING APPLICATIONS				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER FA8650-07-C-7737	
				5c. PROGRAM ELEMENT NUMBER 62303E	
6. AUTHOR(S) Saman Amarsinghe, Robert Miller, and Michael Ernst				5d. PROJECT NUMBER ARPS	
				5e. TASK NUMBER ND	
				5f. WORK UNIT NUMBER ARPSNDBQ	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Massachusetts Institute of Technology 77 Massachusetts Avenue Cambridge, MA 02139-4307				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) <div style="display: flex; justify-content: space-between;"> <div style="width: 45%;"> Air Force Research Laboratory Sensors Directorate Wright-Patterson Air Force Base, OH 45433-7320 Air Force Materiel Command United States Air Force </div> <div style="width: 45%;"> Defense Advanced Research Projects Agency/ Information Processing Techniques Office (DARPA/IPTO) 3701 N. Fairfax Drive Arlington, VA 22203-1714 </div> </div>				10. SPONSORING/MONITORING AGENCY ACRONYM(S) AFRL/Rydi	
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-RY-WP-TR-2009-1033	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.					
13. SUPPLEMENTARY NOTES PAO Case Number: DARPA 14405; Clearance Date: 28 Oct 2009. This report contains color.					
14. ABSTRACT We study the technology innovations required to radically improve the process of understanding and parallelizing performance-critical legacy application code. We demonstrate the usefulness and feasibility of such a system, dubbed Program Reincarnation, using a simple prototype. A Program Reincarnation tool will assist the programmer in replacing the program's code (the body) while preserving the original specification (the soul).					
15. SUBJECT TERMS Legacy Application Code, Program Reincarnation, Parallelization, Multicores, Streaming Applications					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT: SAR	18. NUMBER OF PAGES 158	19a. NAME OF RESPONSIBLE PERSON (Monitor) Alfred J. Scarpelli 19b. TELEPHONE NUMBER (Include Area Code) N/A
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			

Table of Contents

Section	Page
List of Figures	v
List of Tables	v
Acknowledgements.....	vi
1.Executive Summary	1
2.Introduction.....	2
3.Methods, Assumptions, and Procedures	5
3.1 Study of Technology Needs and Innovations	5
3.2 Assisted Application Reincarnation Tool (AART).....	5
3.3 Binary Interpretation and Instrumentation.....	5
3.4 Inference Engine	6
3.5 Streaming Representation	6
3.6 Block Diagram and Specification	7
3.7 Automatic Parallelization.....	7
3.8 Application Study	7
3.9 Workshop on Software Forensic Environments	8
3.10 Prototypes	9
4.Result and Discussions	10
5.Conclusion	11
6. Recommendations.....	12
7.References.....	13
List of Acronyms, Abbreviations, and Symbols.....	14
Appendix A Final Slides from Software Forensic Environment Workshop	15

List of Figures

Figure	Page
1. Design Flow for Program Reincarnation	4
2. Visualized Communication Pattern of MPEG-2 and MP3	7

List of Tables

Table	Page
1. Characteristics of the Parallel Stream Graphs and Performance Results on a 4-Core Machine.....	8
2. List of Software Forensic Environment Workshop Attendees	8

Acknowledgements

We would like to thank our DARPA program manager Dr. William Harrod for identifying the importance of Program Reincarnation, supporting this work and providing numerous insights and guidance during many discussions. AFRL contract management team, led by Alfred Scarpelli, provided ample guidance during the process. Al was especially helpful in reviewing the final report. Thirty industry experts participated in the Software Forensics workshop held at MIT. Their insights and ideas greatly help guide this work and is a major fraction of this report.

1 Executive Summary

As the computing industry matures, it carries with it the immense burden of maintaining the flexibility and performance of decades' worth of legacy code. Legacy programs often have little in common with today's development practices; they were written in different language dialects and targeted a different class of computer architectures. As recent trends demand that modern software be written with explicit parallelism to harness the power of multicore architectures, it is becoming largely intractable to manually upgrade a legacy application to modern performance standards.

We study the technology innovations required to radically improve the process of understanding and parallelizing performance-critical legacy application code. We demonstrate the usefulness and feasibility of such a system, dubbed "Program Reincarnation", using a simple prototype. A Program Reincarnation tool will assist the programmer in replacing the program's code ("the body") while preserving the original specification ("the soul"). Our technique originally focused on streaming applications such as multimedia, graphics, and signal processing; we employ a combination of static and dynamic program analysis to extract the simple, high-level block diagram from the optimized and obfuscated legacy code. Our comprehensive approach is broadly applicable to program understanding, documentation, refactoring, and automatic parallelization.

2 Introduction

Over the last few decades, programmers have written a staggering amount of code. These billions of lines of code have profoundly impacted the human race. Today, programs are everywhere – in computers, cell phone, cars, cameras and cash registers. While computer hardware and technology is going through an incredibly rapid growth period, in many areas computer code seems to enjoy a longevity mainly associated with a mature field. It is not uncommon to actively use computer programs written two decades ago. Furthermore, most new software includes a large amount of program code written long ago. For example, the Microsoft Windows vulnerability MS-03-011 affected Windows 95 to Windows 2003, suggesting that the code written before 1995 was still in use in a program produced 8 years later. In contrast, it is hard to even find a working computer older than seven years, let alone a modern computer built with parts designed two years ago.

This reliance on old code, written using old languages, outdated methods, and targeting now-defunct machines, is creating a massive obstacle to the rapid growth of computers. Most of these legacy programs cannot take full advantage of the exponential performance growth rate of modern processors. Some of the performance-critical legacy programs, ones that were highly optimized for the architecture of the day, will experience slowdowns and compatibility issues on newer processors. Legacy code has had an even larger negative impact in computer architecture. As legacy applications are extremely important, commercial microprocessors have been “wasting” the transistor budgets offered by Moore’s Law to improve the performance of legacy programs at any cost, even if the performance gains are only marginal. Monolithic superscalar processors are a good example of this trend.

Recently, processor vendors have started to break this trend by moving to multicore architectures. Multicore architectures provide much better peak performance per die area, but require programs to be explicitly parallel. This trend, while providing much higher performance to modern programs, puts legacy applications at a disadvantage as they will no longer get performance improvements from new generations of processors.

In order to take advantage of modern multicore processors, one must rewrite these legacy binaries using modern languages and techniques. Apart from performance, there are many other benefits of updating a legacy code base. Most of these programs are written in older languages without the benefit of many powerful language features. Furthermore, during the last decade we have made huge strides in software engineering. Rewriting a legacy code base using these techniques can make them more efficient, malleable, portable, secure, and fault tolerant.

However, rewriting a legacy program is a daunting task. While the application may be in wide use, it is very difficult and time consuming to reverse engineer the exact algorithm implemented and any special cases handled by the application. The original programmers are no longer available in many cases. Information that helped the original programmer such as specification and requirement documents, simulations, mathematical proofs, tests etc. may have been lost. In some cases, even the source code may not be available. Even if the source is available, the tool chain and the libraries have often diverged over the years, making it impossible to build the application from source.

Today, recreating the specification of a legacy application is more error prone and takes longer than the coding task itself. We demonstrate how to drastically reduce the cost of recreating a

program. A Program Reincarnation system provides a tool chain to help the programmer replace the program code ('the body') while adhering to the original program's specification ('the soul'). This tool chain takes advantage of the availability of the source code and a working program to help guide the programmer through the reincarnation process. The DoD modernization effort can hugely benefit from this capability by reducing the software porting cost associated with most hardware upgrades.

Streaming applications amplify the difficulties of the current recreation process while also providing tantalizing possibilities for drastic reduction of programmer effort. Most streaming applications are performance critical. Thus, programmers were forced to hand optimize them for the architecture of the day, making it virtually impossible to understand the underlying algorithm. Furthermore, streaming algorithms do not naturally fit in to old programming models, requiring complicated scheduling and buffer management that further obfuscate the original algorithm. Most of the streaming algorithms were originally developed in prototyping environments such as Matlab. Unavailability of these intermediate representations that helped the original programmer further complicates the extraction of the underlying algorithm. However, most of the streaming algorithms correspond to simple block diagrams with minimal control flow. There are very few special cases in streaming algorithms. Thus, once the underlying algorithm is discovered, it leads to a simple and relatively straightforward representation.

The overall design flow in a comprehensive system for program reincarnation is given in Figure 1. The prototype implementation studied here includes the minimal subset necessary to demonstrate end-to-end flow. The design flow is controlled by the Assisted Application Reincarnation Tool. First, the execution profile of the instrumented binary as well as static analysis of the source code is fed to an inference engine. This engine builds an application knowledge base with the annotated streaming representation of the program. This knowledge base is used for many tasks. The streaming block diagram and the derived specifications are presented to the programmer. In addition, the programmer is provided with hints on refactoring and domain specific transformations of the program. When possible, the system will attempt to automatically parallelize the application. Appropriate tests are generated to help discover the program invariants as well as check the compliance of the reincarnated application.

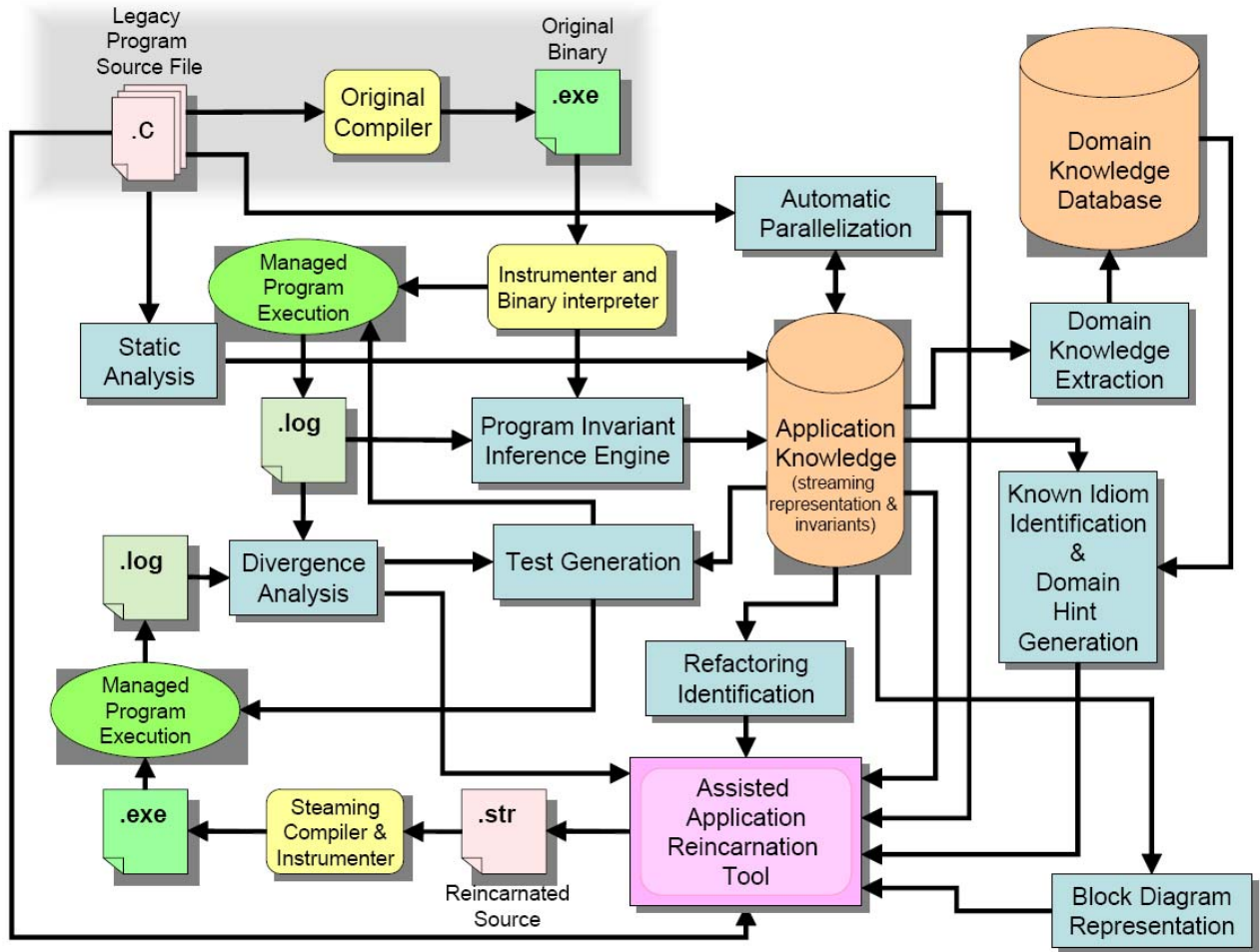


Figure 1. Design Flow for Program Reincarnation

3 Methods, Assumptions, and Procedures

In this section, we identify and investigate the technologies needed for a program reincarnation tool.

3.1 Study of Technology Needs and Innovations

We investigated methods that use sophisticated static analysis of source code, however, we found that information from static analysis is not sufficient. This is mainly due to the complexity of C program and obfuscation by hand optimizations performed by the users. We could not find any existing compiler that was able to extract useful high level information from the complex C programs in our benchmark set.

We have investigated the structure and capabilities of using an application knowledge base. We identified the application information required for program reincarnation and identified an application knowledge base format.

We also studied the technology innovations needed for test generation to support program reincarnation. For example, dynamically detected invariants reveal the properties of the program's execution over the test suite. Such information can be used in feedback-directed random testing generation. In addition, the inferred invariants can also be used as a type of coverage metric such as for test selection and prioritization; more coverage yields a better test suite. These test generation and evaluation techniques enhance the soundness of dynamic analysis and help programmers have better understanding of the legacy code for program reincarnation.

We also studied program refactoring opportunities.

3.2 Assisted Application Reincarnation Tool (AART)

AART is the nerve center of program reincarnation. We demonstrated the feasibility of AART by showing that our simple annotations can easily be added by the programmer. We created a process for extracting coarse grained stream data flow from existing C programs to parallelize these programs by taking advantage of streaming parallelism. We also developed a “global view” of the program behavior that can be extracted from existing programs.

3.3 Binary Interpretation and Instrumentation

The binary interpretation and instrumentation is critical for gathering the invariant information. We study the extensions required for current binary instrumentation tools such as DynamoRIO, PIN and Valgrind. We developed the binary interpretation and instrumentation tool using the Valgrind system. This analysis tool can gather the necessary profile information and extract the data dependence patterns from legacy applications. Our tool interprets every program instruction and recognizes which partition it belongs to. We maintain a table that, for each memory location, holds the identity of the program partition that last wrote to that location. On encountering a store instruction, the partition writing to the location is recorded. Likewise, on every load

instruction, a table lookup is performed to determine the partition that produced the value being consumed. Every unique producer-consumer relationship is recorded in a list and outputted at the end of the program, along with the stream graph and communication macros.

3.4 Inference Engine

We believe that many shortcomings of static program analysis and automatic parallelization can be mitigated by observing the computations performed by the application at runtime, and performing machine learning to generalize its observations. The outputs of the generalizations can be the basis of a program specification given to AART. We have built a simple inference engine to determine the pipeline and data parallel sections. It is possible to build a more powerful inference engine using the Daikon system, which is the state-of-art artificial intelligence based system that can dynamically detect likely high-level program invariants. Those program invariants are useful in program understanding and used to infer communication patterns, which are critical to (semi)automatic parallelization and program reincarnation.

3.5 Streaming Representation

In the prototype system we use a simple streaming intermediate representation, loosely based on the MIT StreamIt compiler. While our primary focus was on streaming applications, we also studied the source code of five open-source Java projects. We analyzed qualitatively and quantitatively the change patterns that developers have used in order to retrofit concurrency. We found out that retrofitting concurrency is not a one time event, but it is a continuous process. The first motivation for retrofitting concurrency is often to increase the responsiveness, and then later the throughput of an application. As the application matures and makes more use of concurrency, the predominant changes fall into fixing concurrency errors, fine-tuning, and improving the scalability. Given the importance and the length of such transformations, tool developers should consider (semi)automation for each stage in the concurrency lifecycle in order to improve programmer productivity.

Many application domains have a rich set of domain specific idioms, program representation standards and program transformation opportunities. For the domain of streaming applications, the steady-state communication pattern is regular and stable, even if the program is written in a language such as C that resists static analysis. We employ a dynamic analysis to trace the communication pattern between program partitions, which is used to construct a stream graph for the application as well as detailed list of producer-consumer instruction pairs, both of which aid program understanding and help track down any problematic dependences.

3.6 Block Diagram and Specification

We have built a tool that can display the parallel regions and the communication pattern in a programmer friendly block diagram representation. Examples of stream graphs for MPEG-2 and MP3 appear in Figure 2. The stream graph presents a coherent high-level block diagram of the application to the programmer.

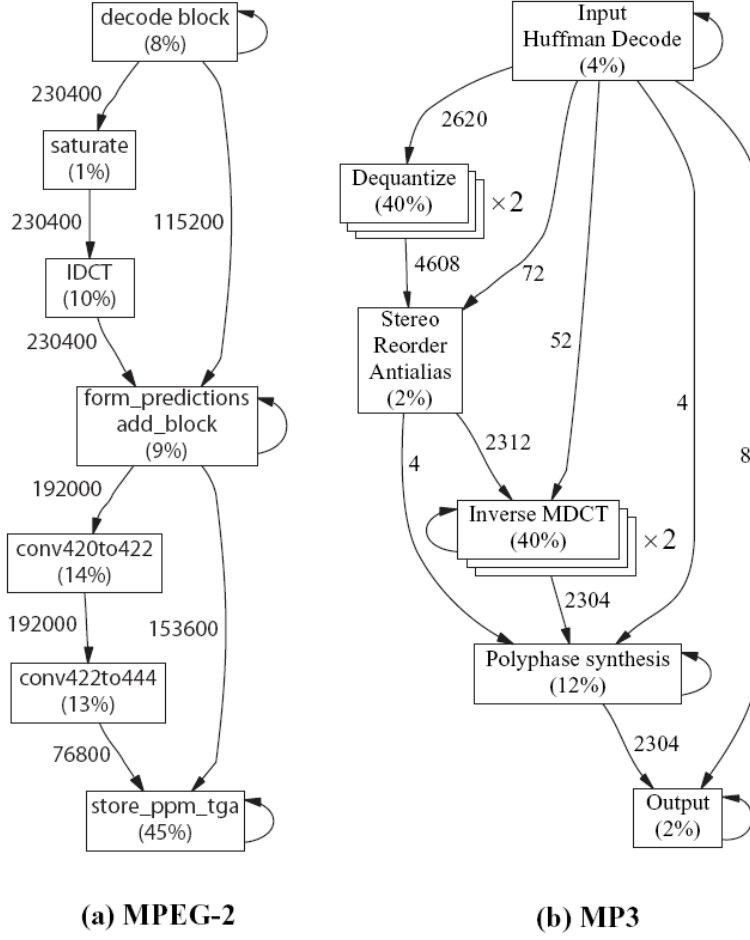


Figure 2. Visualizations Generated by the AART That Shows the Communication Pattern of MPEG-2 and MP3

3.7 Automatic Parallelization

Automatic parallelization is a critical component in this process. If automatic parallelization is successful, it will drastically reduce the work required by the programmer. However, decades of intense research have not achieved fully automatic parallelization. The tool we built performs a partial automatic parallelization. Using the streaming representation, the program is decomposed into distinct execution threads and mapped to a multicore architecture. The prototype employs lightweight programmer annotations, directed by the tool, to achieve a semi-automatic mapping.

3.8 Application Study

We have shown that our tool can extract parallelism out of six real life legacy programs. Three of these are traditional stream programs (MPEG-2 decoding, MP3 decoding, GMTI radar processing), and three are SPEC benchmarks (parser, bzip2, hmmer). The characteristics of the generated parallel stream graphs and performance results on a four-core machine are shown in Table 1. Our analysis extracts a useful block diagram for each application, and the parallelized

versions offer a 2.78x mean speedup on a 4-core machine. Speedups range from 2.03x (MPEG-2) to 3.89x (hmmer).

Table 1. Characteristics of the Parallel Stream Graphs and Performance Results on a Four-core Machine

Benchmark	Pipeline Depth	Data-Parallel Widths	Speedup
GMTI	9	---	3.03x
MPEG-2	7	---	2.03x
MP3	6	2,2	2.48x
197.parser	3	4	2.95x
256.bzip2	3,2	7	2.66x
456.hmmer	2	4	3.89x
GeoMean			2.78x

Data-parallel width refers to the number of ways any data-parallel stage was replicated.

3.9 Workshop on Software Forensic Environments

We held a workshop to discuss the software forensic environment – legacy code reuse, abstraction and representation, portability, and parallelization – concepts at MIT on February 27th 2008. The workshop was attended by the following people:

Table 2. February 27, 2008 Workshop Attendees

Saman Amarasinghe	MIT		Ras Bodik	Berkeley
Bill Harrod	DARPA		Regina Barzilay	MIT
Jon Hiller	STA		Vivek Sarkar	Rice
Robert Miller	MIT		Ralph Weischedel	BBN
Dawson Engler	Stanford		George Heineman	WPI
David Padua	UIUC		Bill Thies	MIT Student
Guang Gao	Delaware		Vikram Chandrasekhar	MIT Student
Una-may O'reilly	MIT		Jason Ansel	MIT Student
Doug Post	HPC		Marek Olszewski	MIT Student
Rick Pancoast	Lockheed		Michael Gordon	MIT Student
Craig Rasmussen	LANL		Danny Dig	MIT
Cornell Wright	LANL		Milissa Benincasa	BRSC
Bob Chambers	Northrop		Michael Van De Vanter	SUN
Alfred Scarpelli	AFRL		Daniel J. Quinlan	LLNL
James Anderson	Lincoln			

Doug Post, Craig Rasmussen, Cornell Wright, and Bob Chambers described the legacy code problems their respective institutes have faced in the past. After a description of ideas from the software forensics environment study (the Reincarnation of Streaming Applications study) three breakout groups worked on a problem description that a potential software forensics environment research effort would attempt to solve; a technology roadmap to support software forensics

environment research and associated innovation and development; and software forensics environment milestones that could be the basis for research and development and used to evaluate technical progress. After further discussion this information was refined and a final set of slides was prepared by the groups (See Appendix A). The workshop highlighted the importance of the legacy code issues and why we think we will be able to solve this problem.

3.10 Prototypes

We have build a simple prototype that can instrument an existing C program, extract data movement by executing the program, analyze the data movement to extract the streaming data patterns, report these patterns to the user using a graphical interface, add annotations to parallelize the stream components and finally parallelize the program.

4 Result and Discussions

In order to demonstrate the feasibility of understanding and extracting the underlying parallelism from legacy code, we implemented an end-to-end system that takes existing legacy C programs and, with minimal programmer help, extracts the parallelism. We focused on streaming applications such as video, audio, and digital signal processing, which are often described in documentation by a block diagram with a fixed flow of data.

To exploit pipeline parallelism using our system, the programmer annotates the natural boundaries of pipeline partitions, and then our system records all communication across those boundaries during a training run. This communication trace is converted to a stream graph that shows the high-level structure of the algorithm as well as a list of producer/consumer statements that can be used to trace down problematic dependences. If the programmer is satisfied with the parallelism in the graph, he recompiles the annotated program against a set of macros that are emitted by our analysis tool. These macros serve to fork each partition into its own process and to communicate the recorded locations using pipes between processes.

We have applied our methodology to six case studies: MPEG-2 decoding, MP3 decoding, GMTI radar processing, and three SPEC benchmarks. Our tool was effective at parallelizing the programs, providing a mean speedup of 2.78x on a four-core architecture. Despite the potential unsoundness of the tool, our transformations correctly decoded ten popular videos from YouTube, ten audio tracks from MP3.com, and the complete test inputs for GMTI and SPEC benchmarks.

5. Conclusion

To summarize, this work makes the following contributions:

- We have shown that for the class of streaming applications, pipeline parallelism is very stable. Communication observed at the start of execution is often preserved throughout the program lifetime, as well as other executions. While the code can be complicated, the underline communication pattern is simple and is amenable to extraction.
- We have defined a simple API for indicating potential pipeline parallelism in the program. Comparable to threads for task parallelism or OpenMP for data parallelism, this API serves as a fundamental abstraction for pipeline parallelism.
- We developed a dynamic analysis tool for tracking producer/consumer relationships between coarse-grained program partitions. The tool outputs a stream graph of the application, which validates or refutes the parallelism suggested by the programmer. It also provides a detailed statement-level trace and a set of macros for automatic parallelization.
- We applied our methodology to six case studies, encompassing MPEG-2 decoding, MP3 decoding, GMTI radar processing, and three SPEC benchmarks. We extracted meaningful stream graphs of each application, and achieve a 2.78x mean speedup on a four-core architecture.

6. Recommendations

It is clear that legacy program code is an extremely important issue for US competitiveness and national security. The conclusion of the software forensic environment workshop is that the advent of multicore and other technologies will make the legacy problem even more acute as applications will be forced to restructure because of these technologies. Thus, finding a solution to the legacy problem is of great national importance.

We observed that, today, we are closer to finding a viable technological solution for this problem. However any viable solution will have to combine aspects of multiple emerging technologies in different areas such as program understanding, analyses and compilation, program verification and testing, and human interfaces. This will require a substantial effort and will need to bring together researchers from these separate communities.

7. References

A Practical Approach to Exploiting Coarse-Grained Pipeline Parallelism in C Programs. William Thies, Vikram Chandrasekhar, Saman Amarasinghe. International Symposium on Microarchitecture. Chicago, IL. December, 2007.

“The Daikon system for dynamic detection of likely invariants” Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao.

Science of Computer Programming, vol. 69, no. 1--3, December, 2007, pp. 35-45.

“How do programs become more concurrent? A story of program transformations” by Danny Dig, John Marrero, and Michael D. Ernst. MIT Computer Science and Artificial Intelligence Laboratory technical report MIT-CSAIL-TR-2008-053, (Cambridge, MA), September 5, 2008.

“Refactoring sequential Java code for concurrency via concurrent libraries” by Danny Dig, John Marrero, and Michael D. Ernst. MIT Computer Science and Artificial Intelligence Laboratory technical report MIT-CSAIL-TR-2008-057, (Cambridge, MA), September 30, 2008.

“Feedback-directed random test generation” by Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. In *ICSE'07, Proceedings of the 29th International Conference on Software Engineering*, (Minneapolis, MN, USA), May 23-25, 2007, pp. 75-84.

List of Acronyms, Abbreviations, and Symbols

Acronym	Description
AART	Assisted Application Reincarnation Tool
DoD	Department of Defense
MPEG-2	MPEG-2 video decoder, M(oving) P(icture) E(xperts) G(roup) - 2 is a standard for coding of moving pictures and associated audio information
MP3	MP3 audio decoder, MP3 (MPEG-1 Audio Layer 3) is a digital audio encoding format
GMTI	Ground Moving Target Indicator
bzip2	Compression and decompression algorithm
hmmer	Calibrating HMMs for biosequence analysis
HMM	Hidden Markov Model
SPEC	Standard Performance Evaluation Corporation

SFE Workshop

- 8:00 – 8:30 Breakfast**
- 8:30 – 10:05 Description of the Legacy problem from the DOD/DOE user community**
- 8:30 – 9:00 Doug Post, HPC
- 9:00 – 9:30 Craig Rasmussen and Cornell Wright, LANL
- 9:30 – 9:55 Bob Chambers, Northrop Grumman
- 9:55 – 10:05 Rick Pancoast, Lockheed Martin
- 10:05 - 10:15 Break**
- 10:15 - 11:00 SFE: current thinking and plan of action (Bill & Saman)**
- 11:00 - noon Breakout working groups**
- Problem Description (D407)
- Technology Roadmap (this room)
- Milestones and Evaluation (D451)
- noon - 1:00 Lunch**
- 1:00 - 2:00 Breakout working groups continued**
- 2:00 - 3:30 Feedback from the working groups**
- 3:30 - 3:45 break**
- 3:45 - 4:45 General discussion**
- 4:45 - 5:00 Wrap-up**

SFE: Software Forensics Environment

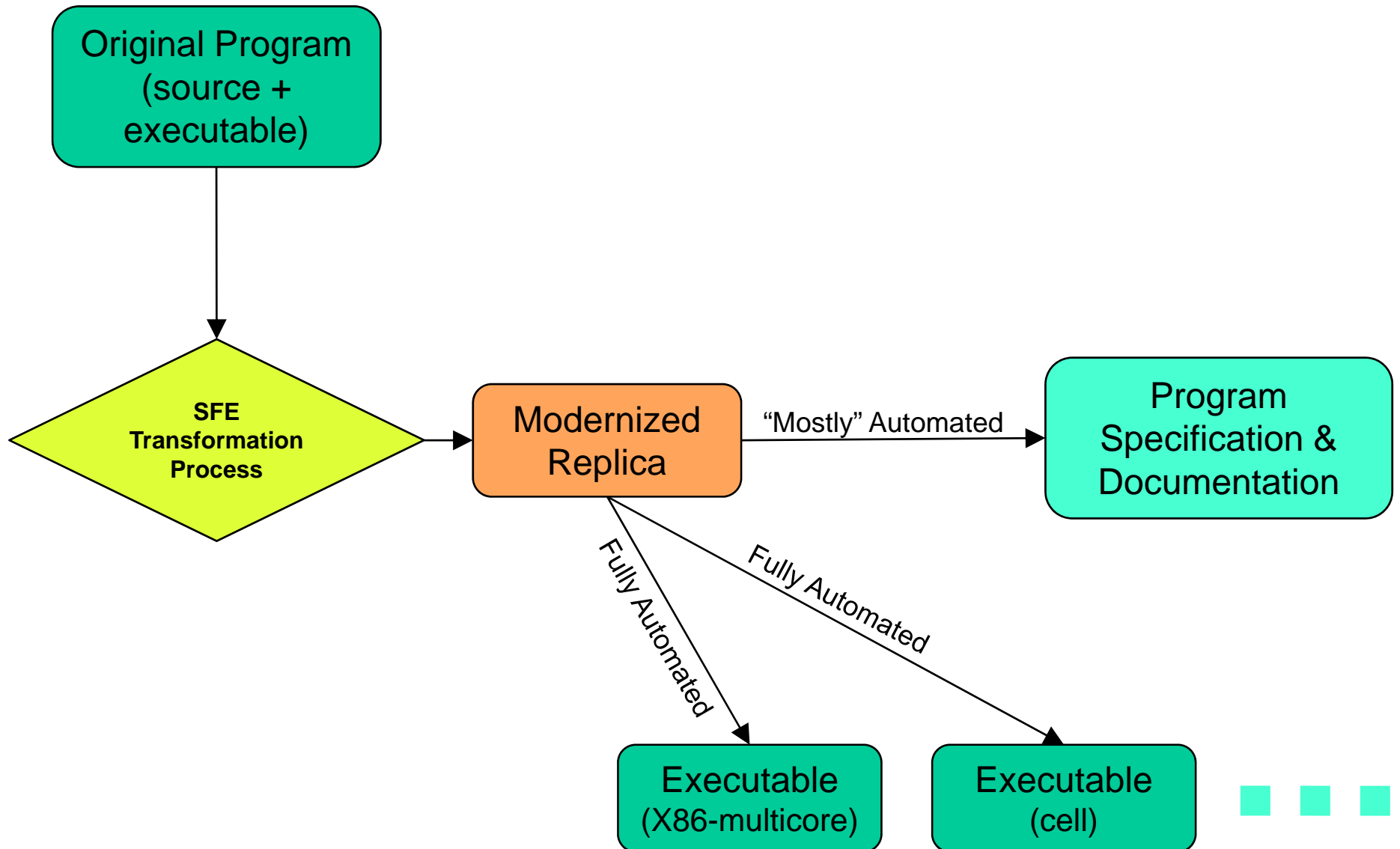
Legacy Code

- **310 billion lines of legacy code in industry today**
 - 60-80% of typical IT budget spent re-engineering legacy code
 - (Source: Gartner Group)
- **Even a bigger problem for the DoD**
 - Lifetime of systems are much longer in DoD
 - Mission critical code were highly optimized for the original machine
- **Now code must be migrated to multicore machines**
 - Current best practice: manual translation

SFE: Software Forensics Environment

- **Solve the problem of modernizing the legacy code base**
- **Create a program within DARPA IPTO to spearhead the innovation of necessary technology**
- **The main idea of the program**
 - Invent the technologies needed to transform a legacy program into a “modernized replica”
 - The modernized replica behave identically to the original program
- **Goals of the program**
 - The cost of transformation is greatly reduced (by 100x to 1000x)
 - Errors and deviations of the transformed program is reduced (eliminated?)
 - The modernized replica can be trivially mapped in to multiple current architectures
 - Modernized replica can take advantage of multicore and other forms of parallelism in modern architectures.
 - Modernized replica is easy to understand and manage

SFE Outline

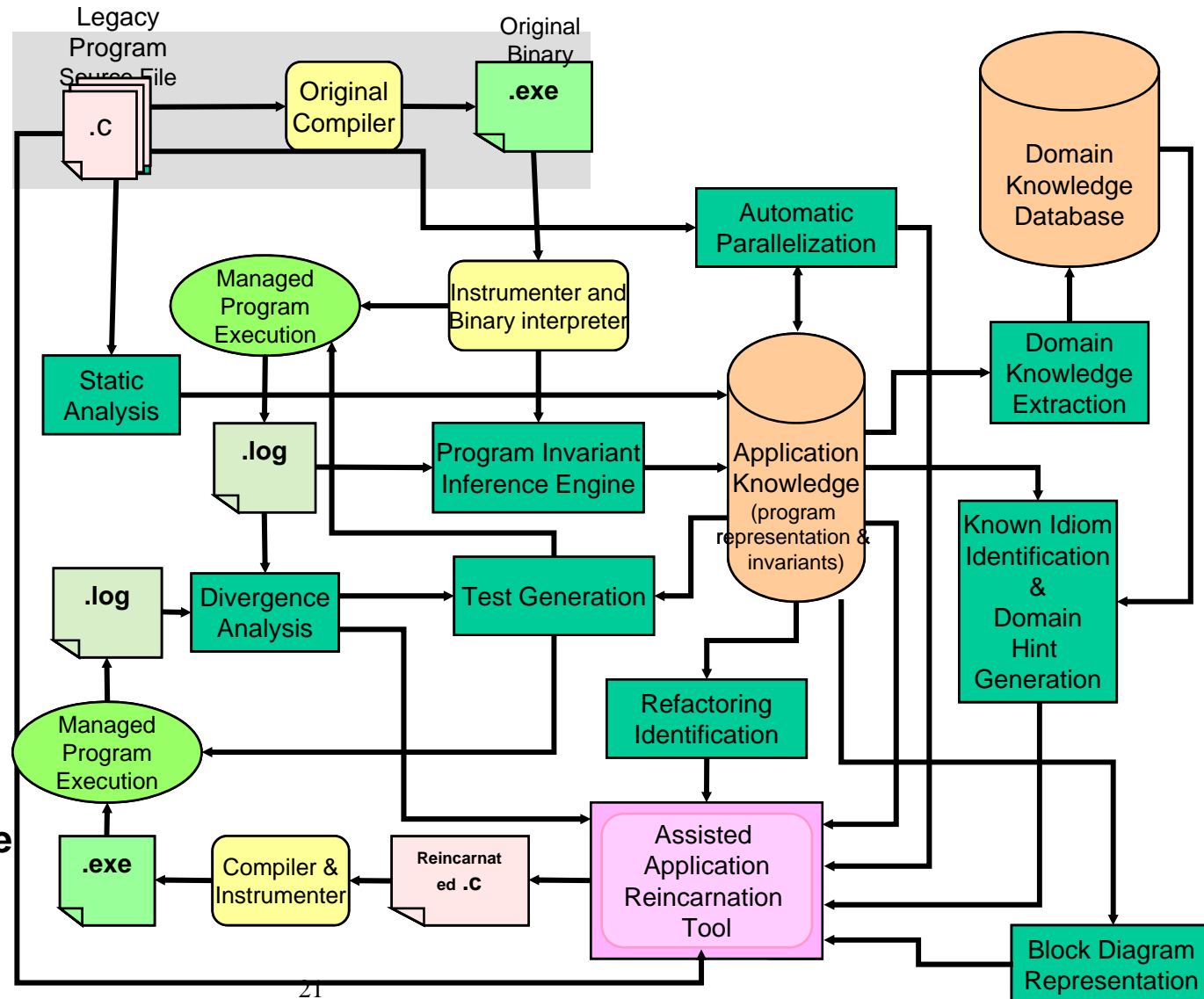


SFE Transformation Process

- **Start with the existing program**
 - Assumes that we have
 - Compilable/buildable source
 - Working executable + test suite
- **Produce a Modernized Replica**
- **Holistic process involving tools and programmers**
 - Each team proposes a process
 - The process can involve
 - Static program analysis and model checking
 - Dynamic analysis and testing
 - Tools to assist the programmer (HCI)
 - Learning
 - Automated and assisted program transformation
 - Programmer education and training

Program Reincarnation: A Holistic Approach

- **Dynamic analysis**
 - Managed program execution
 - Program invariant inference
 - Application knowledge database
- **Assisted parallelization**
 - GUI tool
- **Correctness in reincarnated**
 - Test Generation
 - Divergence Analysis
- **Static analysis**
 - Automatic parallelization
 - info for program understanding
- **Learn about the domain**
 - Flag domain specific issues
 - Generate domain-specific hints
- **Bring programs to modern age**
 - Block diagram
 - Refactoring identification



Modernized Replica

- **Functionally equivalent to the original program**
 - Original program as the specification
- **Convertible to an executable form**
 - Preferably an automated path to multiple platforms
- **Will provide the following benefits**
 - Ability to easily create an executable for multiple modern computer systems
 - Ability to extract performance from modern computer systems
 - multi-processor, multicore and/or SIMD parallelism
 - new memory systems
 - Restructured to make it adhere to modern software engineering practices
 - Exposes the high level structure and provide better documentation to make it more programmer-comprehensible
 - Simplify maintenance and provide the capability to expand the program.
- **SFE Support many choices (each teams propose one)**
 - Examples: StreamIt, Matlab, etc.

Breakout Sessions

- **Small working groups (6 to 12)**
- **In each group appoint**
 - A Moderator
 - A Scribe
- **In the first session**
 - Try to define what should be done in the program
 - Bring questions and discussion points on cross-cutting issues to lunch
- **Working lunch**
 - Discuss some of these issues (1 slide summary by each scribe)
- **In the second session**
 - Get into the detailed description (specification)
- **In the general discussion session**
 - Each group gets 20 minutes to present

I Problem Description

- **A clear and crisp description of the legacy problem**
- **What should be the scope of SFE?**
- **Should SFE be confined to a single:**
 - Class of programs and/or Input language
 - IDE
 - Language of the modern replica
- **Domain of the legacy code**
 - FORTRAN, C, MPI, or chosen by the proposer?
 - Signal processing, simulations or chosen by the proposer?
 - Should we select a set of benchmarks?
 - Programs with both a legacy version and a hand modernized version?
- **Types Modernized Replica**
 - Leave it for the teams to propose a format?
 - Matlab, C with libraries, StreamIt etc.
- **Room D407**

II Technology Roadmap

- **What technologies should be part of SFE?**
- **For each technology**
 - Why is it not done today?
 - What can be reasonably achieved during the program?
 - How can that drastically improve the SFE process?
- **Examples**
 - Use unsound techniques, give “mostly sound” suggestions to the user
 - Use the working program as a prototype
 - static and dynamic techniques to generate a “specification”
 - Help test the modernize replica for the compliance with the working program
 - Use natural language processing type techniques to identify similar/interesting parts in a large code base
 - Use learning to identify repetitive and frequent tasks and provide hints to the user
- **This Room (Star)**

III Milestones and Evaluation

- **Reduced cost (by 100x to 1000x)**
 - If the process is fully automated → trivial
 - Programmer involvement → user studies?
 - Can we find a set of applications with original legacy version and a version modernized using current practices that can be used as the base case?
- **Reduction of errors and deviations**
 - What is a good measurement?
- **Modernized replica trivially map in to multiple modern architectures**
 - “trivially map”: Automated tools, no programmer intervention
 - “multiple modern architectures”: at least one distributed memory multicore (cell or Tile64) and one shared memory multicore (core 2 duo or niagara)
- **Modernized replica is efficient and effective**
 - Show speedups against the original program on that architecture
 - Show scalability from one core to max number of cores available
- **Modernize replica can be easy to understood and managed**
 - How to measure quality of the specifications created?
 - How to measure malleability and extendibility of original vs. modernized replica?
- **Room D451**

Breakout Session Assignments

- I Problem Description (D407)
- II Technology Roadmap (this room)
- III Milestones and Evaluation (D451)

	AM	PM			AM	PM
Saman Amarasinghe	roam	roam		Vivek Sarkar	M&E	TR
Bill Harrod	roam	roam		Ralph Weischedel	TR	TR
Jon Hiller	M&E	M&E		George Heineman	TR	TR
Robert Miller	M&E	M&E		Bill Thies	PD	TR
Dawson Engler	TR	TR		Vikram	M&E	TR
David Padua	PD	PD		Jason Ansel	TR	PD
Guang Gao	PD	TR		Marek Olszewski	TR	M&E
Una-may O'reilly	TR	TR		Michael Gordon	M&E	TR
Doug Post	PD	M&E		Danny Dig	TR	PD
Rick Pancoast	PD	PD		Milissa Benincasa	PD	PD
Craig Rasmussen	PD	PD		Michael Van De Vanter	M&E	TR
Cornell Wright	TR	M&E		Daniel J. Quinlan	TR	TR
Bob Chambers	M&E	M&E		Al Scarpelli	PD	PD
James Anderson	TR	M&E		Regina Barzilay	TR	
Ras Bodik	TR	TR	27			



SFE Problem Description



What is the Problem?



- **Billions of dollars are currently invested in operational systems**
 - National Security and Economic Interests exist to sustain the operational capabilities of these systems
 - Software lifetime for these systems is 20+ years
- **The lifetime of hardware installations for current DOE Systems is 18 – 36 months**
 - The legacy software applications must run on the next generation architecture as well as previous versions
- **Currently DOE legacy software is ported to the next generation architecture by hand**
 - This is expensive and a time intensive process
 - Estimated current cost per line of code to rewrite the code is \$100.00, average lines of code for a single DOE application is a half a million. Therefore the cost for one application is 50 million dollars.

MANUAL PROCESS IS ERROR PRONE!



What is the Objective?



- **Maintain and extend DOE flagship applications in support of National Security interests and Economic competitiveness**
 - Need the capability of transforming these applications to run on new hardware architectures
 - Need tools/techniques to:
 - Reduce errors in porting/transforming the application
 - Reduce application development costs
 - Improve application code maintainability
 - Keep acceptable performance



Program Approach



- **Define high-level program abstraction which will allow the transformation of legacy applications**
- **Develop semi-automated mechanisms to transform legacy applications into the modern replica**
- **Define and create analysis tools to aid humans to understand and preserve the knowledge contained in the legacy application**
- **Develop tools to ensure program correctness throughout the transformation process**
- **Develop tools to assist in the maintenance of transformed programs**

SFE Workshop

**IC Conversion Issues into the new ODOE
Environment**

Robert Chambers

Business Development Enterprise Architect

Mission Systems

Northrop Grumman Corporation

Overview

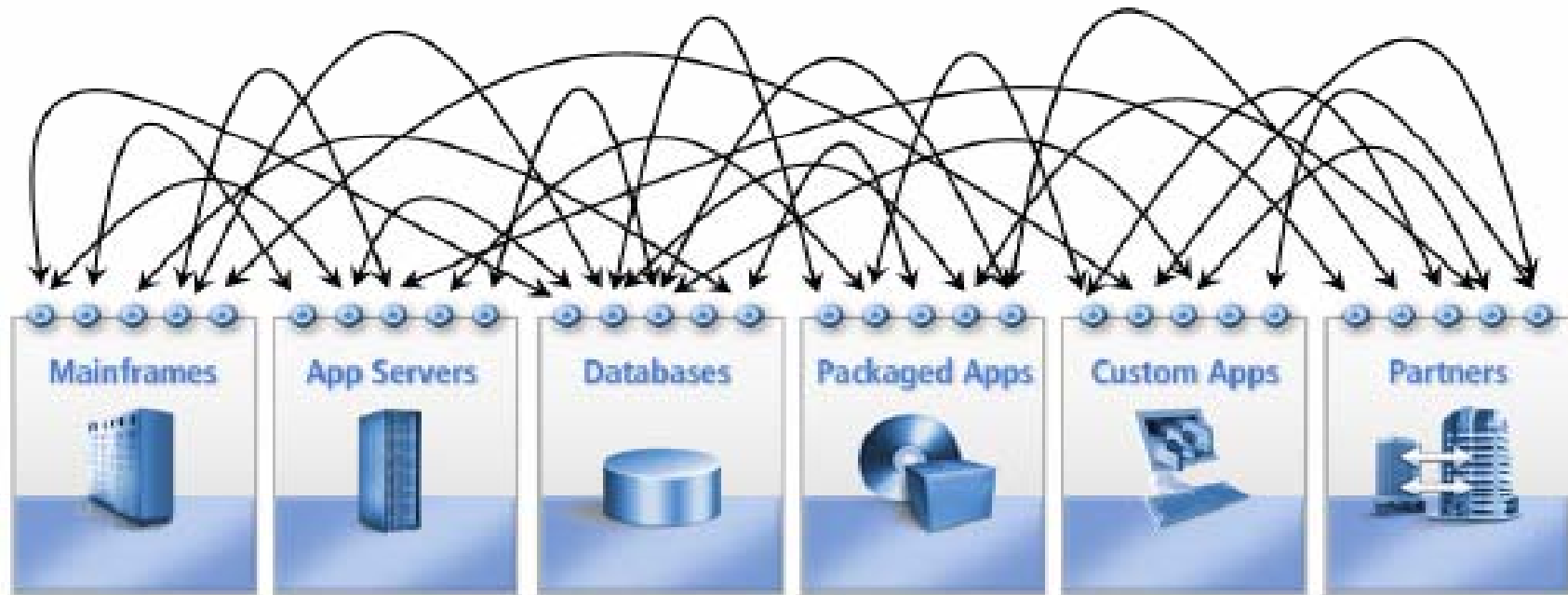
- **IC Transformation Overview**
- **Overview of current processing paradigm**
 - Explain current architecture
- **Director of National Intelligence (DNI) Framework**
- **New Driving requirements**
- **Evolution vs. Revolution**
 - How to convert monolithic stove pipe APIs into component services that leverage the new technologies
- **Discussion**

IC Transformation Overview

- **The intelligence community (IC) is in the process of revolutionizing its computing infrastructures to exploit innovations in emerging system paradigms, such as service-oriented and event-driven architectures.**
 - This development permits consolidation and integration of diverse systems to dramatically increase intelligence asset value and enable new levels of autonomic cross-intelligence collaboration and system intelligence.
 - The goal is to produce intelligent autonomic systems that are virtual, event driven, and secure in a globally distributed environment.
- **The unique challenges in servicing the future IC are**
 - Continual High-bandwidth data ingest rates with low latency timing requirements
 - Complex-event processing (CEP)
 - Knowledge management and retention
 - Intelligence analyst process consolidation and automation
 - Intelligent network development
 - Fielding intelligent secure networks

Extending the SOA model out to 2020 adds the following new capabilities:

- **A development environment that is fully service-oriented with complete registry integration and governed via integrated workflow management.**
- **Security that is included from service concept to service retirement as an integral part of the SOA.**
 - Security becomes a set of services within the SOA.
- **An autonomic on-demand operating environment that is self-aware and self-managing.**
 - It load-balances and optimizes the infrastructure to meet service contracts and quality-of-service agreements on the fly.
- **High-performance grid computing with high-speed, externalized, shared-memory buses between computing nodes with high redundancy and failover.**
- **An intelligent network with intelligent application-aware routers that route global messaging traffic for optimal performance.**
- **Services that can dynamically configure themselves to meet objectives as in grammar-oriented architectures.**



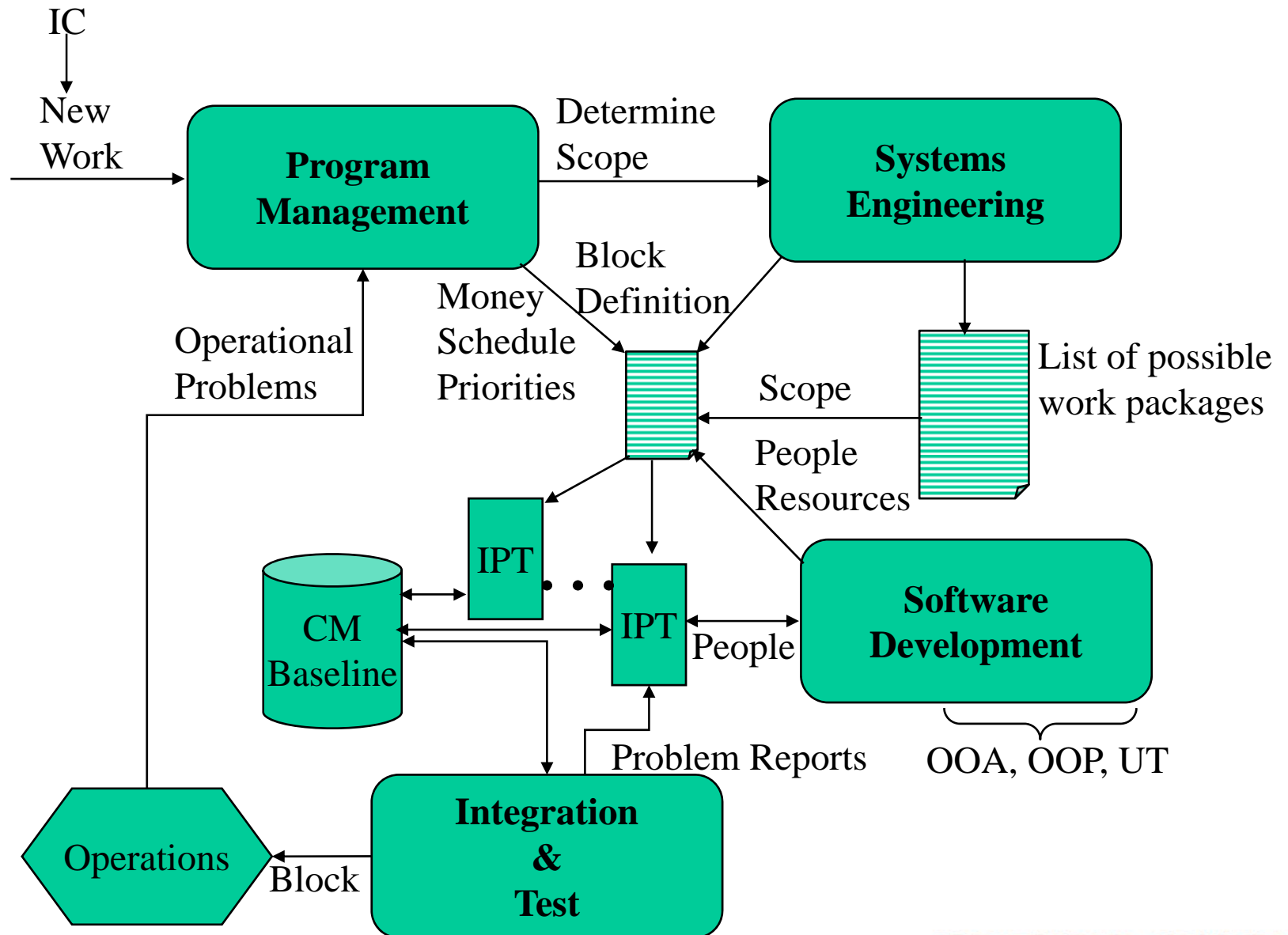
Current IC Architectural Overview

“The Stovepipe Paradigm”

Current IC Processing Paradigm

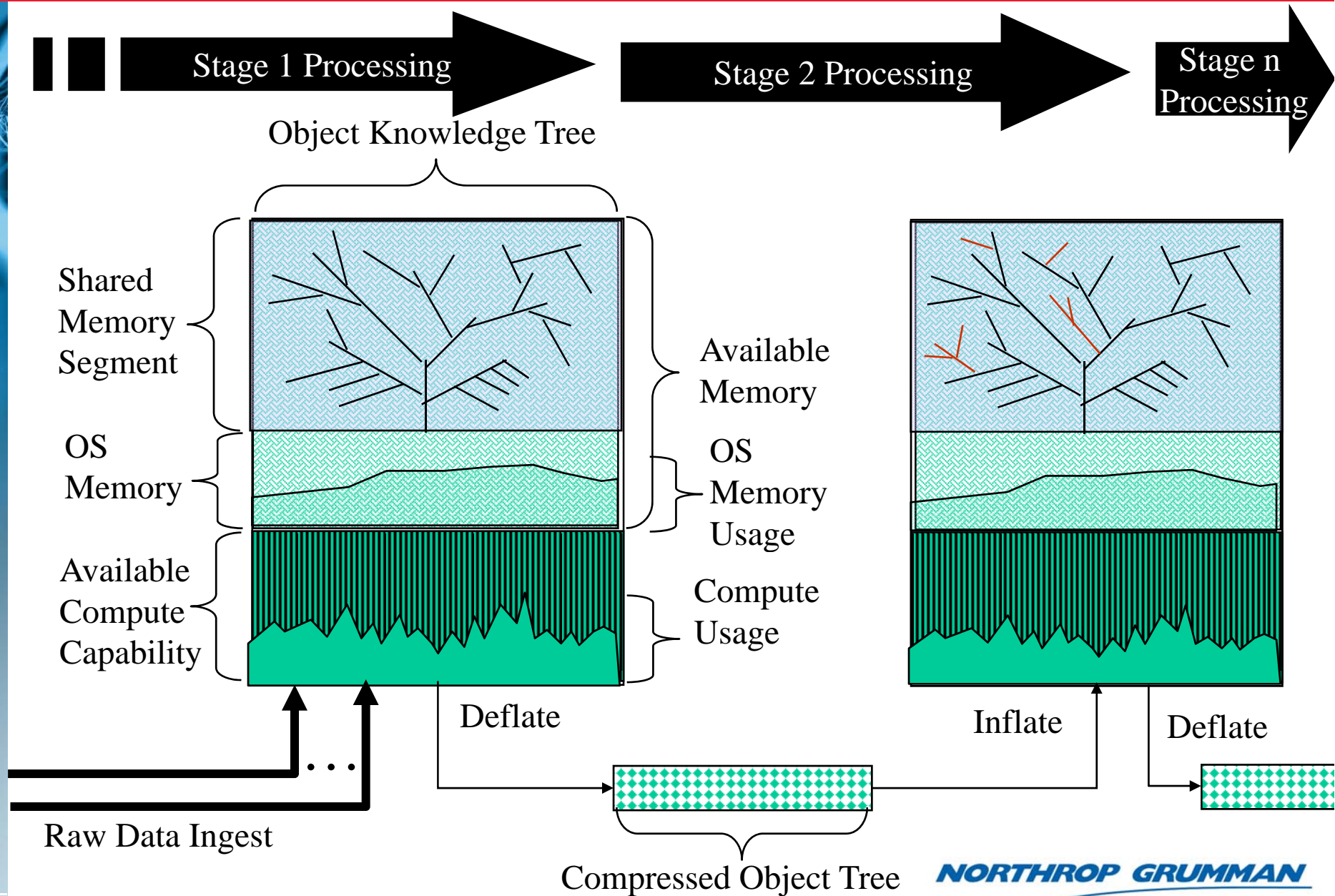
- **Hundreds of Megabytes/Sec to Gigabytes/Sec continually (24X7X365)**
- **Many Petabytes of Storage**
 - SAN and NAS
 - Structured (Database)
 - Unstructured (Flat files with Metadata)
- **Product timeliness in the minutes**
- **Very I/O Intensive algorithmic monolithic processing**
 - Extensive use of SMP Virtual Memory
 - Super Computers packed full of CPUs to get enough memory to build the Virtual Memory
 - Memory managed through project developed mechanisms
- **Hundreds of Systems**
- **Millions of lines of code (Fortran, C, C++, JAVA)**

Example Current Process



Processing Example

(Adding a New Branch to the object requires "Everyone" to change!)

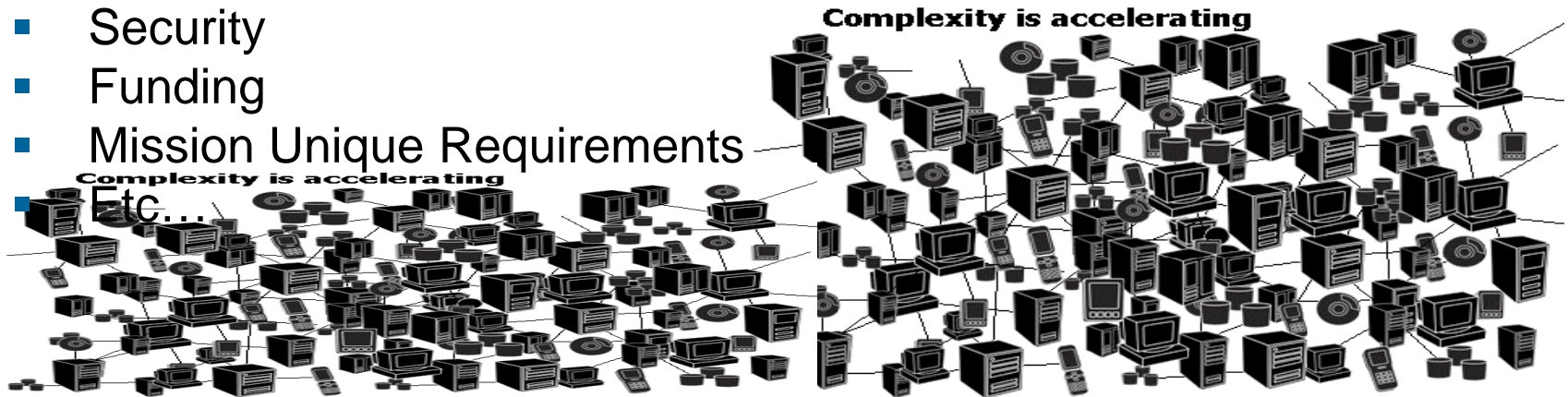


NORTHROP GRUMMAN

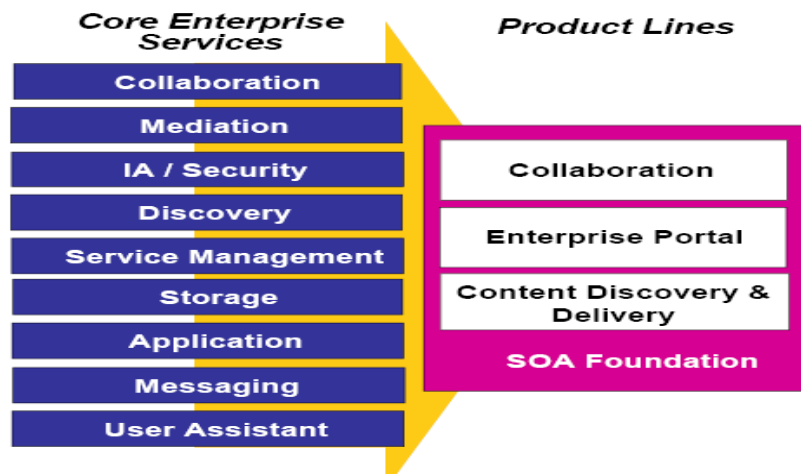
New Driving requirements

- The IC Stove Pipe architecture resulted for a number of reasons

- Security
- Funding
- Mission Unique Requirements
- Etc...

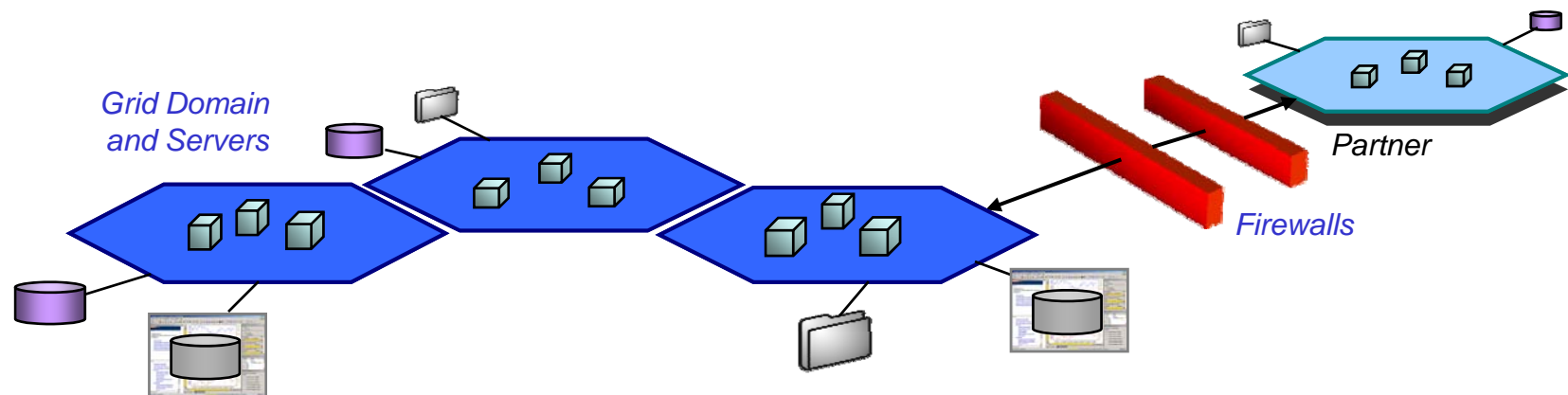


- The result has manifested into Processing Centers that have no floor space, have Power & Cooling issues, and staggering TCO to maintain an ever increasing variety of ageing hardware equipment and software licenses.
- With decreasing congressional funding, less and less money is being allocated to keep up with emerging threat technologies.
- The IC has no choice but to modernize.
- The Federal Enterprise Architecture (FEA) and the Director of National Intelligence (DNI) are the governing bodies for the facilitation of this change
 - <http://www.whitehouse.gov/omb/egov/a-2-EAModelsNEW2.html>
 - <http://www.dni.gov/>



Evolution vs. Revolution

The Chicken or the Egg!



NORTHROP GRUMMAN

How Do We Get There?

How Do We Manage It?

Yesterday

- Systems
- Software interoperability
- Duplication of functions
- Data element standardization
- Common operating environment
- COTS
- Host applications locally
- Heavy clients

Tomorrow

- Service providers and consumers
- Business process integration
- Optimization and specialization
- Standardized business products
- Integrated networked environment
- Commercial service providers
- Provide service globally
- Application delivery over network

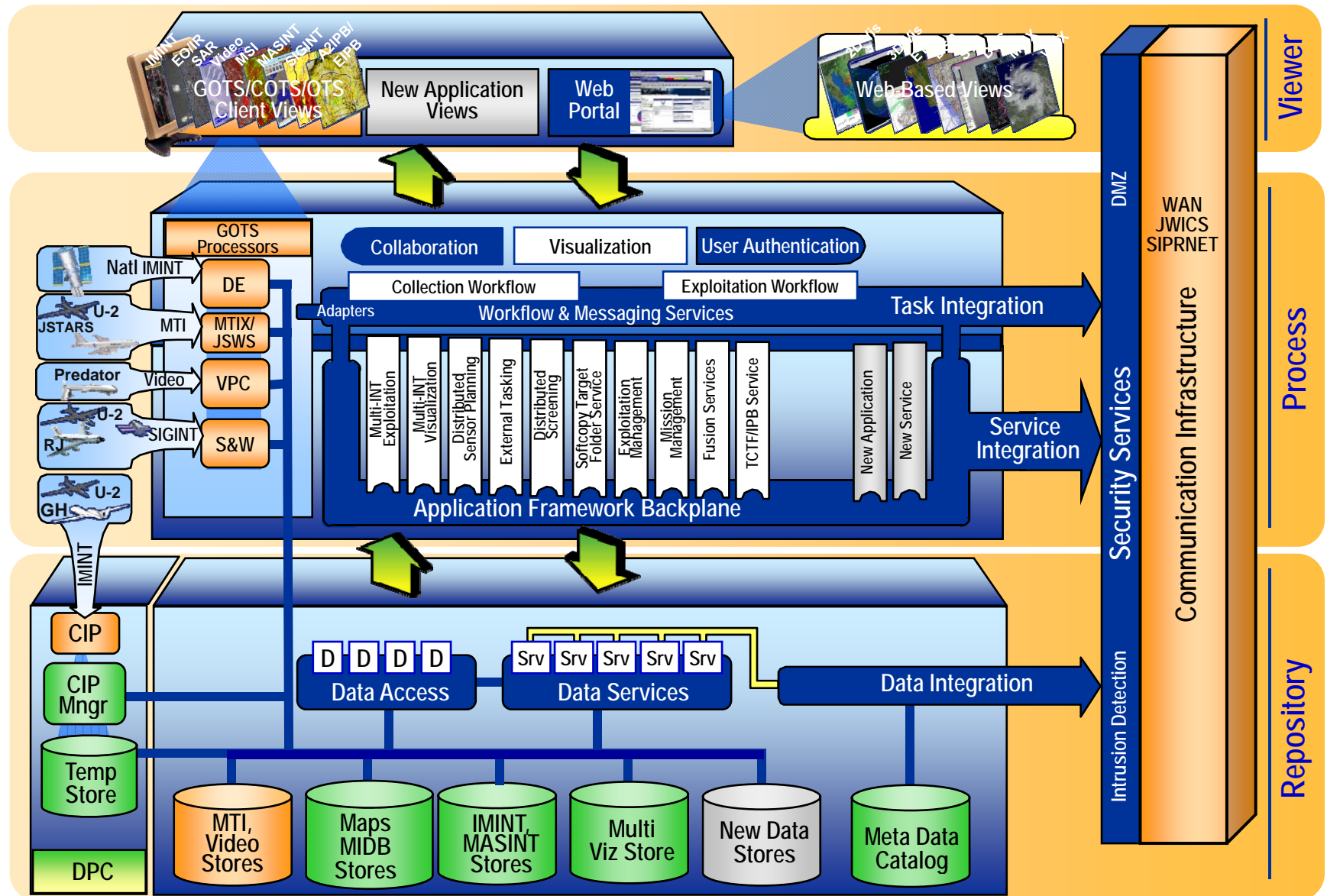


Service Oriented Architecture



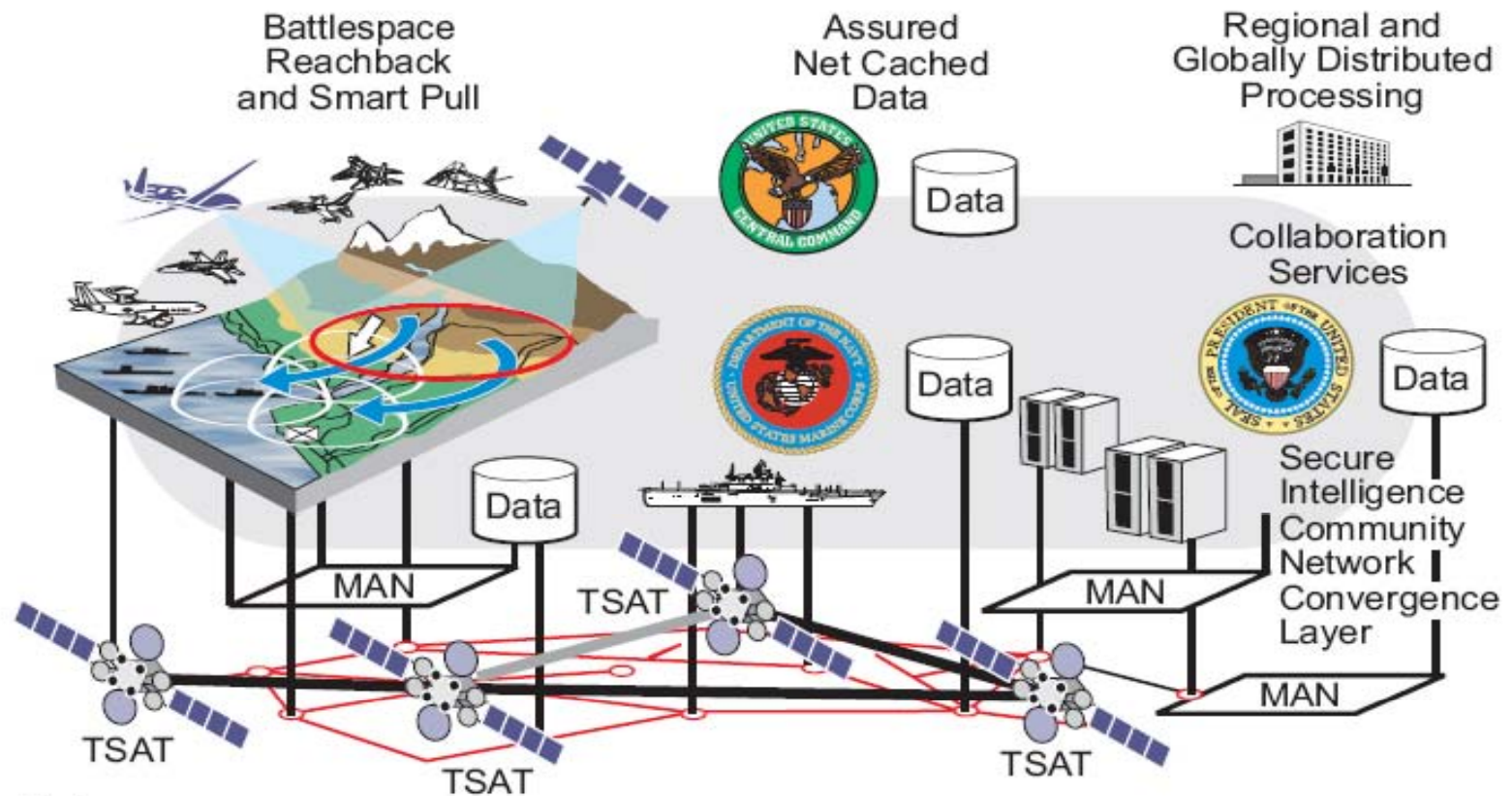
NORTHROP GRUMMAN

Multi-Int Core Reference Architecture



NORTHROP GRUMMAN

Summary



Note:
 MAN = Metropolitan Area Network
 TSAT = Transformational Communications Satellite

The IC has World Class Processing Problems

- **The 2020 IC SOA described here is much more than today's notion of a service-oriented architecture.**
- **The 2020 SOA vision:**
 - Is a service-oriented, event-driven, virtualized, grid computing fabric that is knowledgeable and self-aware
 - Manages the many complexities of advanced computing infrastructures automatically with minimal human intervention
 - Is self-optimizing and self-adjusting to meet the ever-changing needs of the IC enterprise
 - Can be easily expanded and adjusted by humans to encompass new mission needs
- **This revolutionary way of doing business in the IC, will be procured and implemented over the next few years.**
 - Once operational, the IC SOA/grid/EDA will continue to evolve as new technologies are plugged in and played in support of the evolving intelligence environment.

Discussion & Questions



NORTHROP GRUMMAN

DEFINING THE FUTURE

Backup

**Intelligence,
Surveillance and
Reconnaissance**

Systems Integration

Electronic
Systems

Radar and
Air Defense

Satellite
Technology
Information
Technology

Nuclear
Aircraft
Carriers

Unmanned
Systems

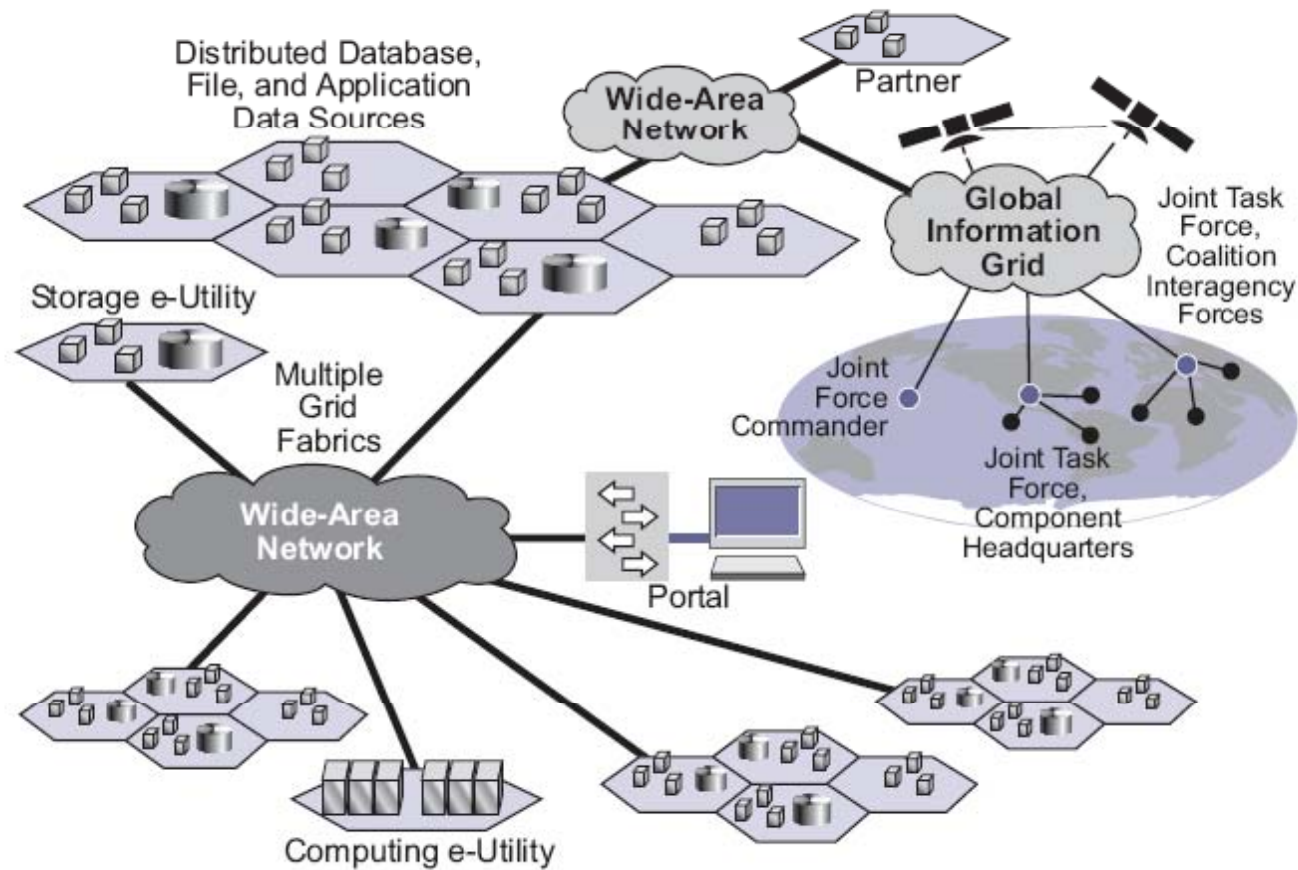
Missile
Defense
Space
Systems

Navigation Systems
Shipbuilding

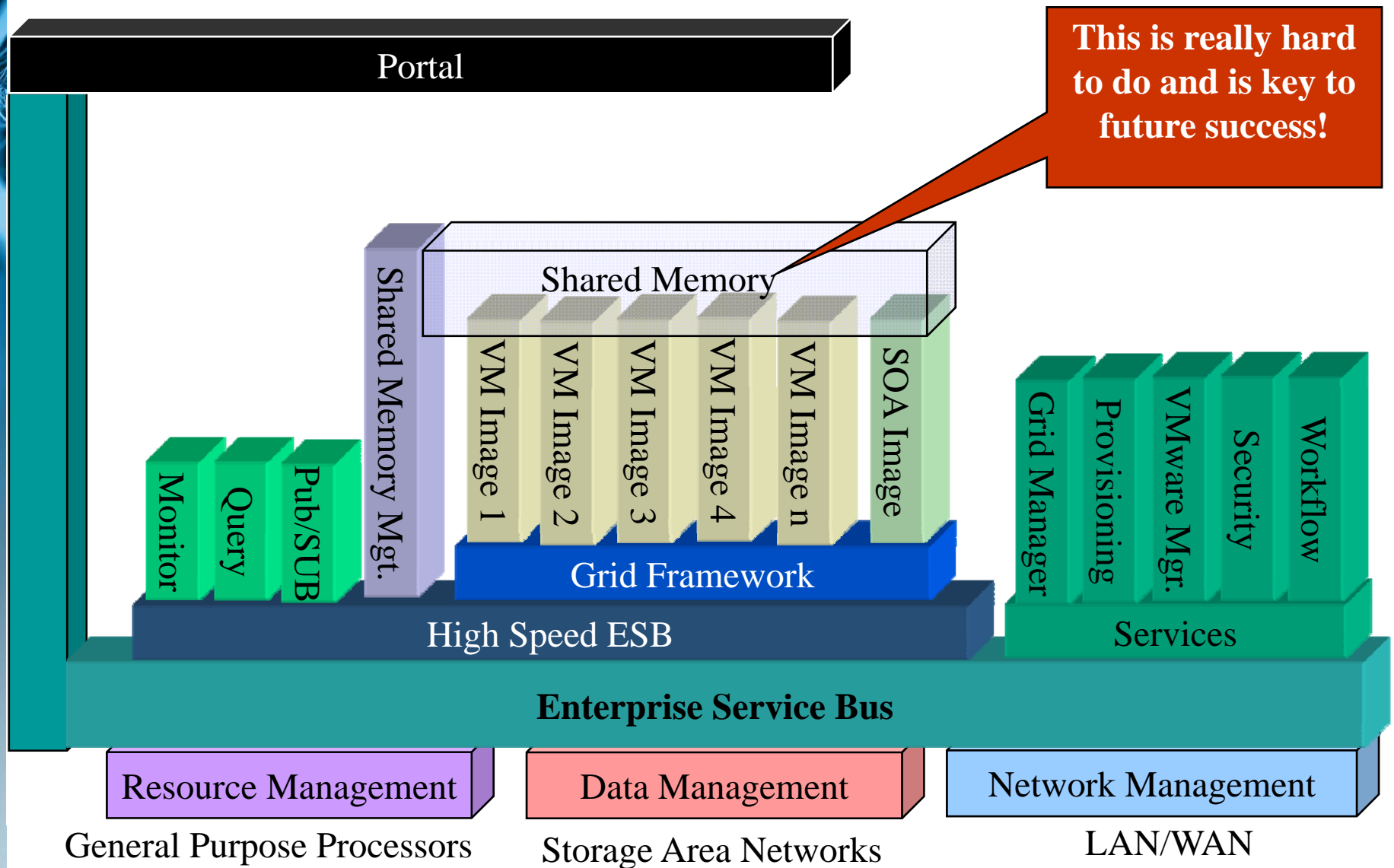
Multi-Stage Transformation is Required

- 1. Build the Revolutionary Infrastructure**
- 2. Wrap Legacy Processing so that it runs as Service Based on the new infrastructure**
- 3. Decompose Legacy Processing into functional elements**
 - Abstract functional elements into encapsulated services available for reuse
 - Model encapsulated services into late binding compound services and generate BPEL
 - Build Workflow Plans to perform an end-to-end thread
 - Test and Validate
- 4. De-Commission Wrapped Legacy API**
- 5. New processing becomes operational**

Technology Needs

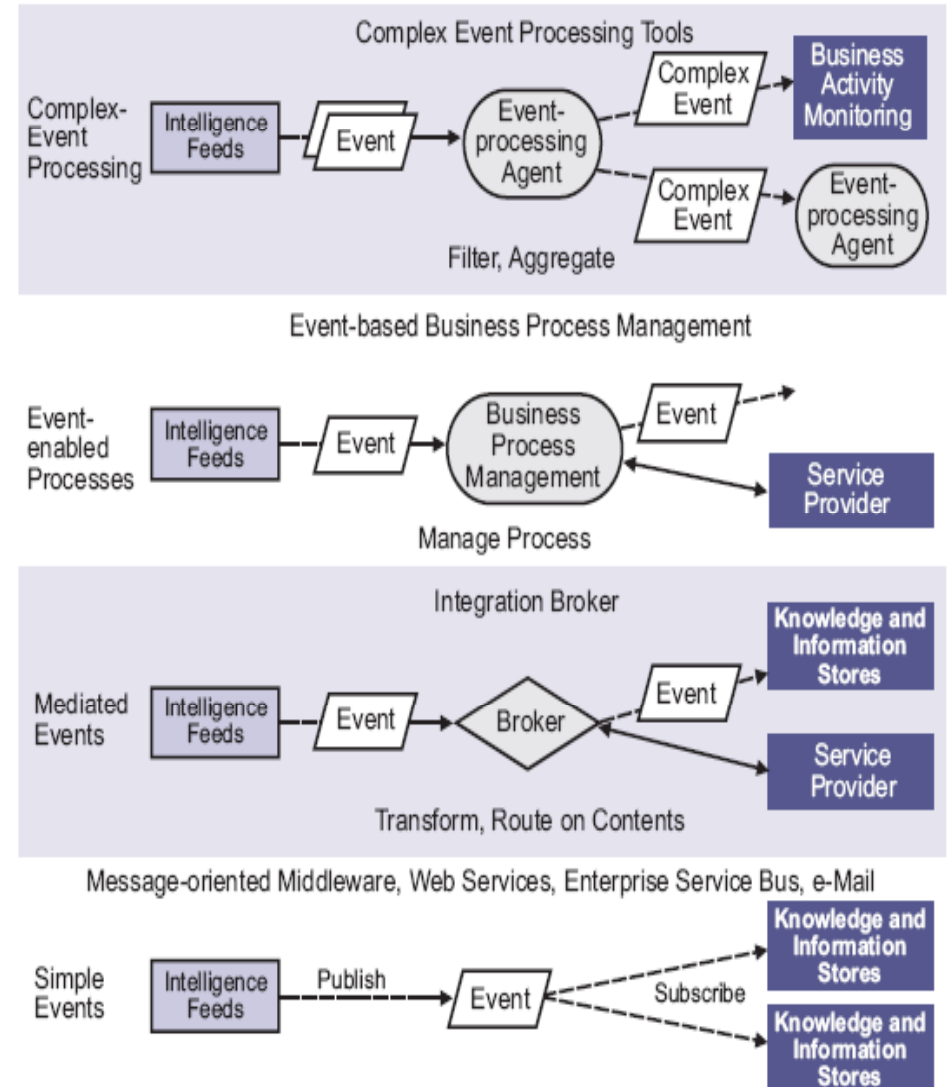


ON DEMAND OPERATING ENVIRONMENT



Event-Driven Applications

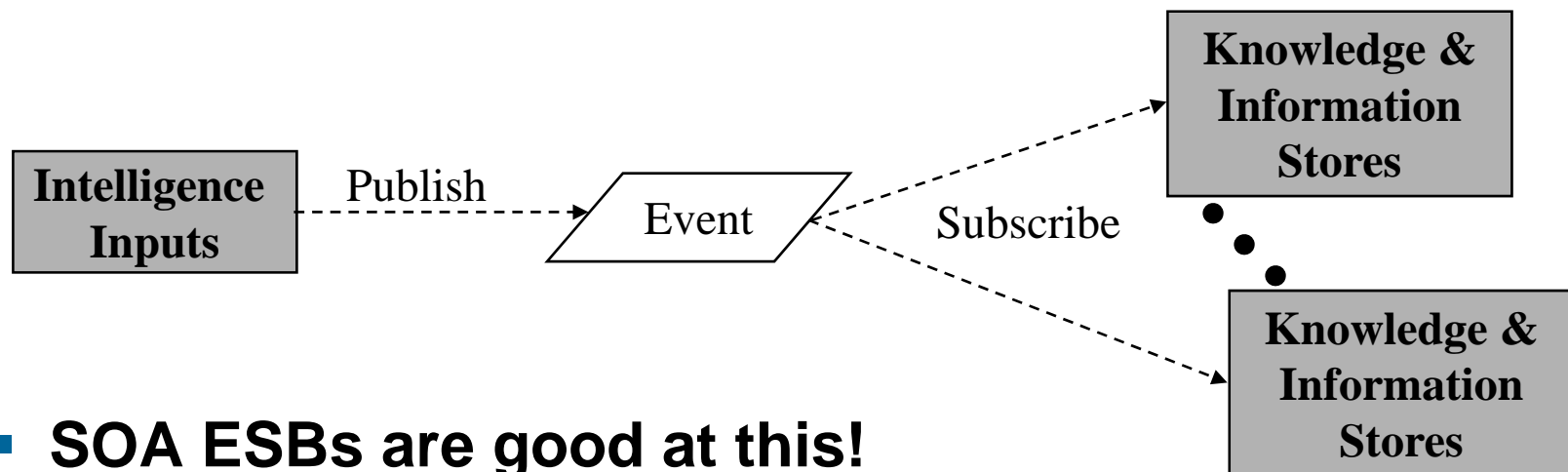
- **Event processing and analysis are critical to the formulation of sound intelligence.**
- **The IC's job is to predict and prevent negative complex events such as those of 9/11.**
 - Two emerging fields EDA and CEP will be an integral part of the IC's 2020 SOA.
- **EDA applications can be sorted into four categories: the first three are aimed at engineering better intelligence systems; the fourth, CEP, is aimed at expanding insight:**



NORTHROP GRUMMAN

Simple Events

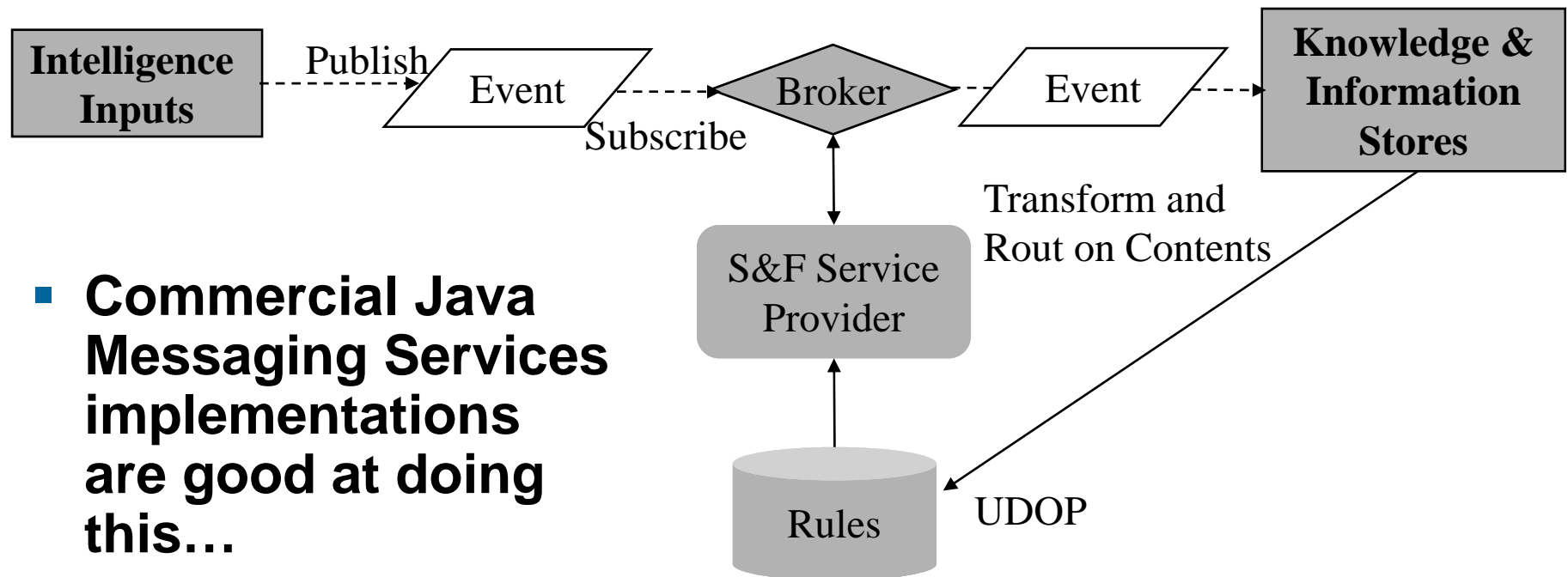
- **Simple EDA, where application programs explicitly send and receive messages directly to and from each other**
 - For example, through message-oriented middleware or Web services.
 - This is the publish and subscribe model.



- **SOA ESBs are good at this!**

Integration Broker

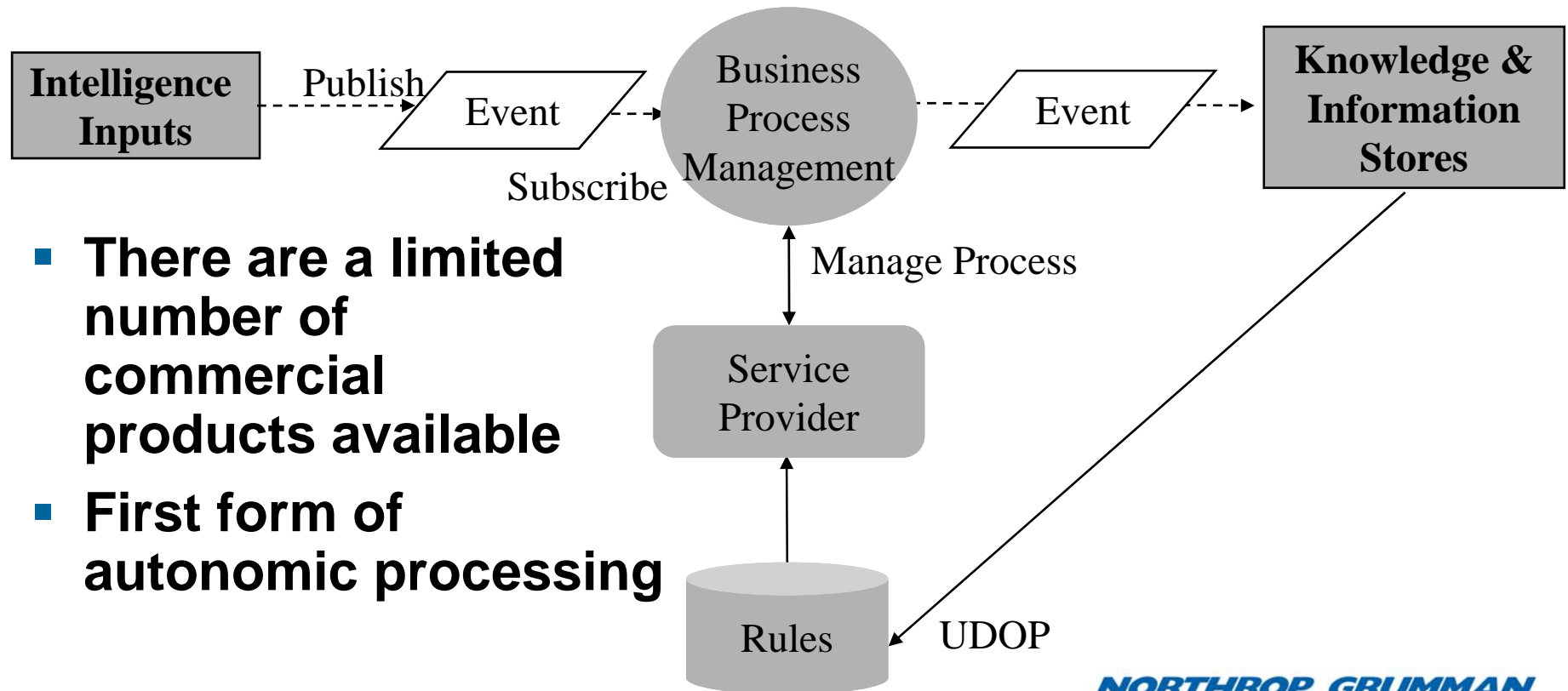
- EDA mediated by integration brokers, which transform and route simple-event messages according to logical rules. This can be viewed as *rule-based event processing*.



- **Commercial Java Messaging Services** implementations are good at doing this...

Event-enabled processes

- EDA directed by business-process-management (BPM) engines, which conduct the end-to-end flow of a multi-step process using BPM-oriented events.



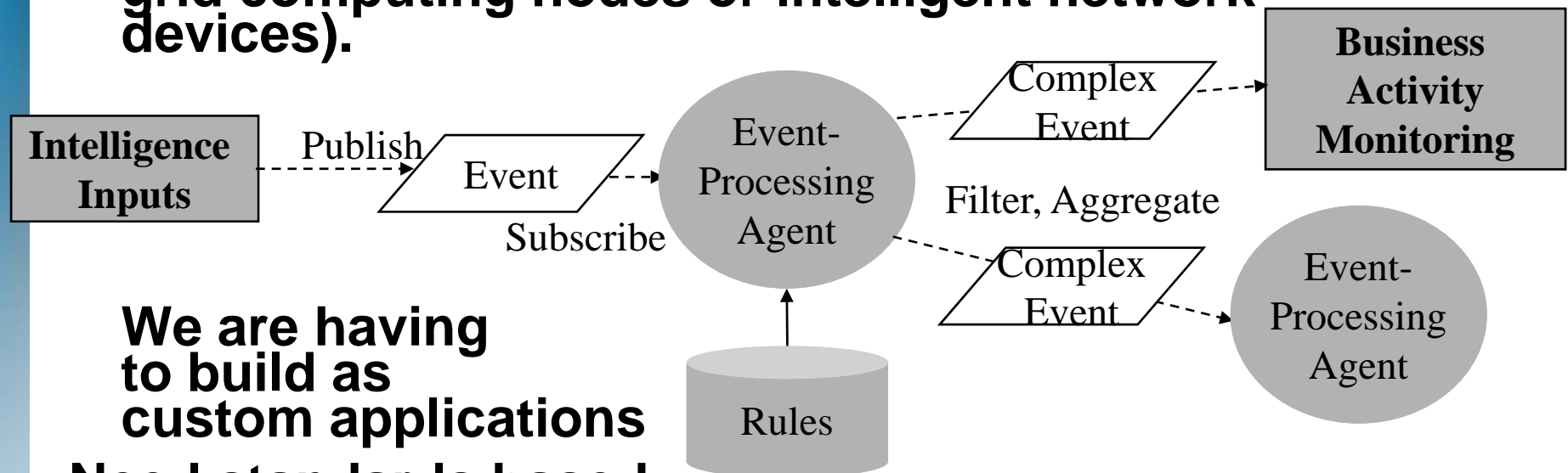
- There are a limited number of commercial products available
- First form of autonomic processing

NORTHROP GRUMMAN

Complex-event processing

This is really hard to do and is key to future success!

- CEP applications, where a sophisticated event manager or network of event-processing agents logically evaluates multiple events from one or more event streams to provide better insight for sense-and-respond applications and business-activity monitoring.
- This type of monitoring is used for signal analysis, security vigilance, and related functions. The processing can occur in any intelligent device (e.g., grid computing nodes or intelligent network devices).



We are having to build as custom applications
Need standards based Autonomic capabilities

SFE: Software Forensics Environment A DoD Systems Integrator Perspective

SFE Workshop

Prepared for the SFE Workshop and DARPA IPTO at MIT

Presented by: Rick Pancoast

856-722-2354

rick.pancoast@lmco.com

Lockheed Martin MS2 - Moorestown, NJ

28 February 2008

Legacy DoD Code

- **Legacy DoD Code Exists in Many Languages, Some Obscure**
 - Today, C and C++ are common
 - Legacy: Ada, Jovial, CMS-2, FORTRAN, etc.
- **TADSTAND C (Tactical Digital Standard, ~ 1990) Mandated all DoD Code be Written in Ada (HOL) or Assembly Language (including CMS-2 [UYKs])**
 - This is why it is the way it is
 - Needed SECDEF dispensation to deviate from the TADSTAND
- **With the DoD Push for COTS and Open Architecture (OA), there has Been a 180° Turnabout**
 - TADSTANDS are no longer invoked

But the “Mess” is Out There

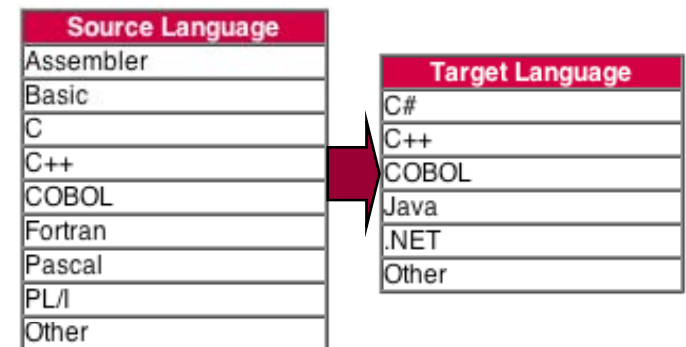
DoD Code Conversion Has Been Done

- **DoD Has Used Code Conversion Software, and Hand Conversion**

- Johns Hopkins APL CMS-2 to Ada Used Experimentally in late 80's
- Command & Control: CMS-2 to C++ (Open C&D); Hand Ada to C++
- Weapons: Ada to Java (hand conversion)

- **Commercial Code Conversion is also Rampant:**

- Java to Visual C# [Microsoft Java Language Conversion Assistant 2.0]
- C to C++ [Free Software Foundation]
- C to VHDL is Popular
- Datatek (Business partner with IBM and Sun)
Provides "Language Conversion Services"
- TSRI, Many Others . . .



- **No One is Really Addressing the Multicore Issue**

- Application Software needs to be Mapped Efficiently to Multiple Processors

**Code Conversion Has Been Used by DoD -
And it Does Work . . . But . . .**

DoD Code Conversion and Validation

- **Code Conversion is Probably the Easy Part**
 - Code Can be Quickly Checked for Proper Functionality
- **The Tough Part is Verifying and Validating the Converted Code - to the Same Pedigree as the Original Code**
 - A Significant Portion of Development Cost is Validation and Verification
 - Regression Testing Can be Very Costly (Error Branches, etc.)
 - Automated Regression Testing (as Part of the Conversion Process) Would be Extremely Valuable
- **SFE Can Provide a Valuable Tool for DoD**
 - Code Conversion (with Multicore - Multiprocessor Target Architecture)
 - Converted Code Verification and Validation
 - Regression Testing

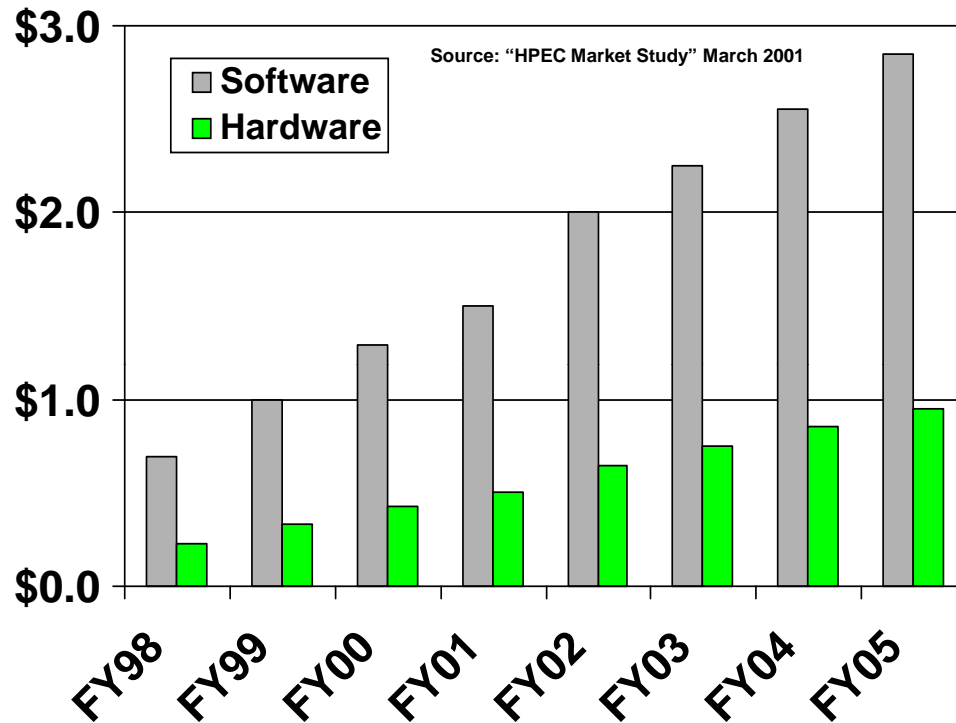
Sample DoD Code Can Be Used for Verification



Why Is DoD Concerned with Embedded Software?



Courtesy of Dr.
Jeremy Kepner,
MIT Lincoln Lab



**Estimated DoD expenditures
for embedded signal and
image processing hardware
and software (\$B)**

- COTS acquisition practices have shifted the burden from “point design” hardware to “point design” software
- Software costs for embedded systems could be reduced by one-third with improved programming models, methodologies, and standards

MIT Lincoln Laboratory



CREATE

Computational Research & Engineering for Acquisition Tools & Environments

To Port Or Not To Port. There Is No Question*.

**Douglass Post – DoD High Performance
Computing Modernization Program**

**Robert Gold – DoD Defense Research and
Engineering**

MIT Computer Science Dept./DARPA Workshop on Code Porting/Reuse

Feb 28, 2008, MIT Computer Science Dept., Cambridge, MA



DoD HPC Modernization Program

HPC Centers



Networking

Defense Research & Engineering Network



HPCMP Resources

ERDC MSRCs
CRC & SMDC ADCs
Users/25 Locations/87 Project
REN Sites
Challenge Projects/2 DHPIs
Institutes/2 Portfolios

HPCMP Resources

MSRC
Users/25 Locations/205 Project
REN Sites
Challenge Projects/4 DHPIs
Institute/2 Portfolios

Force HPCMP Resources

MSRC
CC ADC
Users/36 Locations/190 Project
REN Sites
Challenge Projects/1 DHPI
Institutes/1 Portfolio

Use Agencies

PA, DTRA, JNIC, JFCOM,
, & OTE
Users/4 Locations/11 Project
REN Sites
Challenge Projects/2 DHPIs

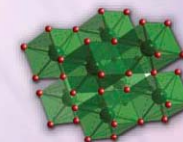
ADC
PI
University

Software Applications Support

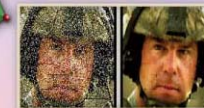
Institutes/Portfolios



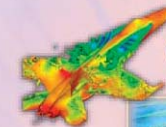
Education & Outreach



PET



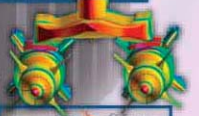
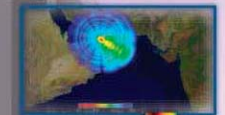
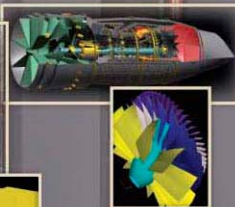
SPI



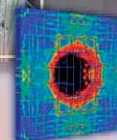
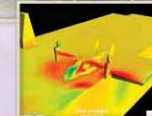
CREATE



DHPIs



Challenge Projects

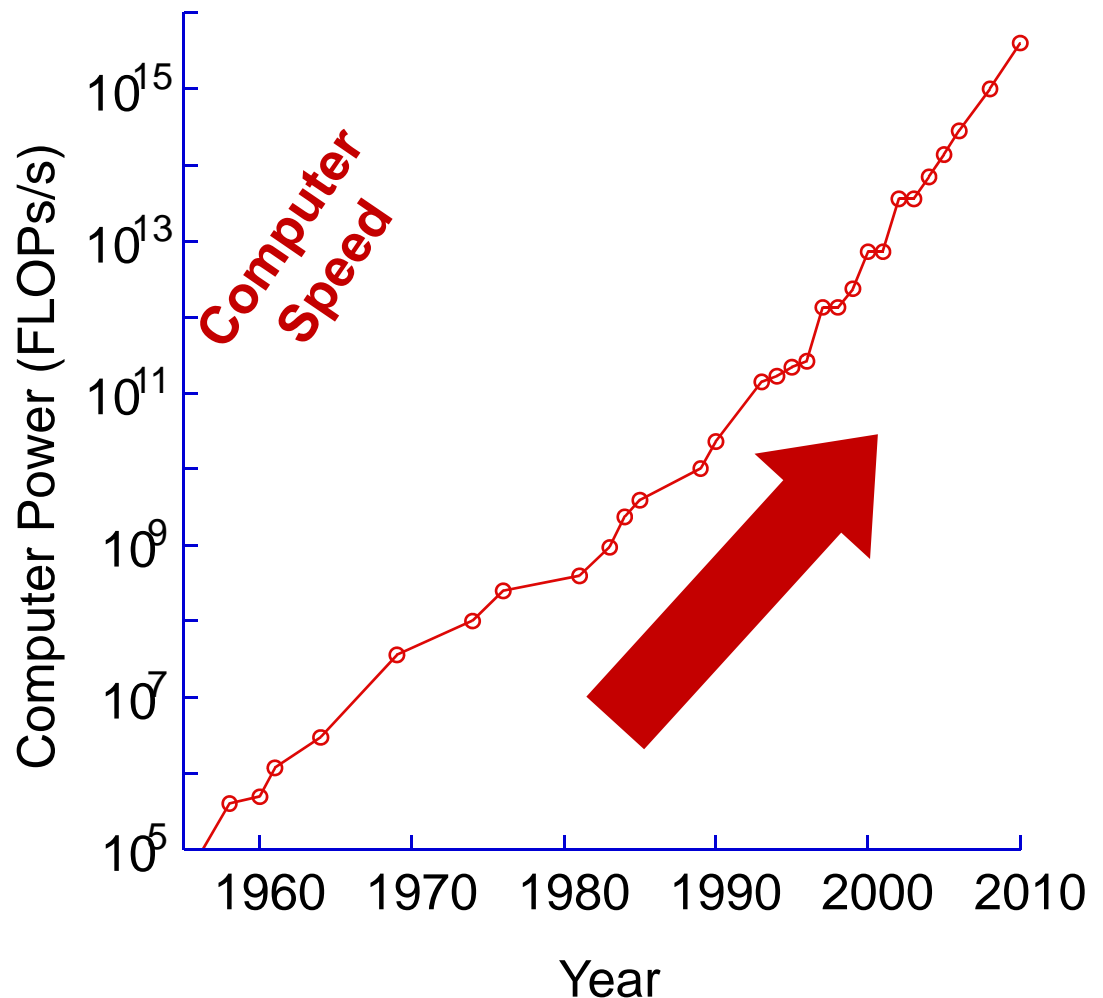




Exponential Growth In Supercomputer Speed And Power Is Making It A “Disruptive” Technology.

Enable paradigm shift

- Potential to change the way problems are addressed and solved
- Make reliable predictions, about the future*
- Superior engineering & manufacturing
- Enable research to make new discoveries
- A vastly more powerful solving methodology!



Computer power comes at the expense of complexity!



The future is exa-flops/s (10^{18} Flops)

Extrapolation to 2020
(1-10 GFlops/core)

2000: 7.2 TFLOPs/s

~5000 cores

2010: 2×10^3 TFLOPs/s

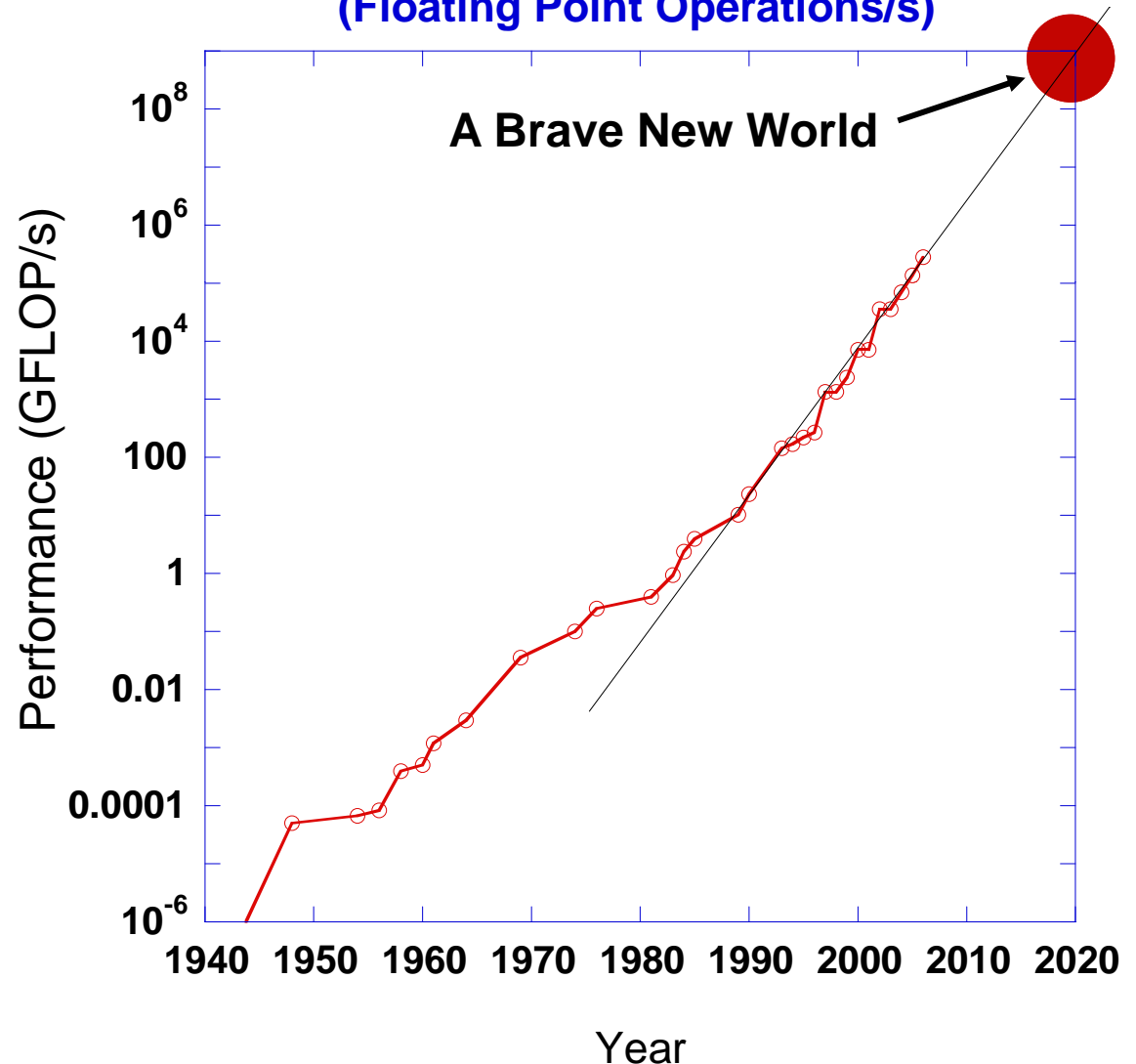
10^{5-6} cores

2020: 10^6 TFLOPs/s

10^{8-10} cores

How do we program
for 10^{8-10} cores?
Especially if the
cores are different?

Computing Power
for the world's fastest computer
(Floating Point Operations/s)





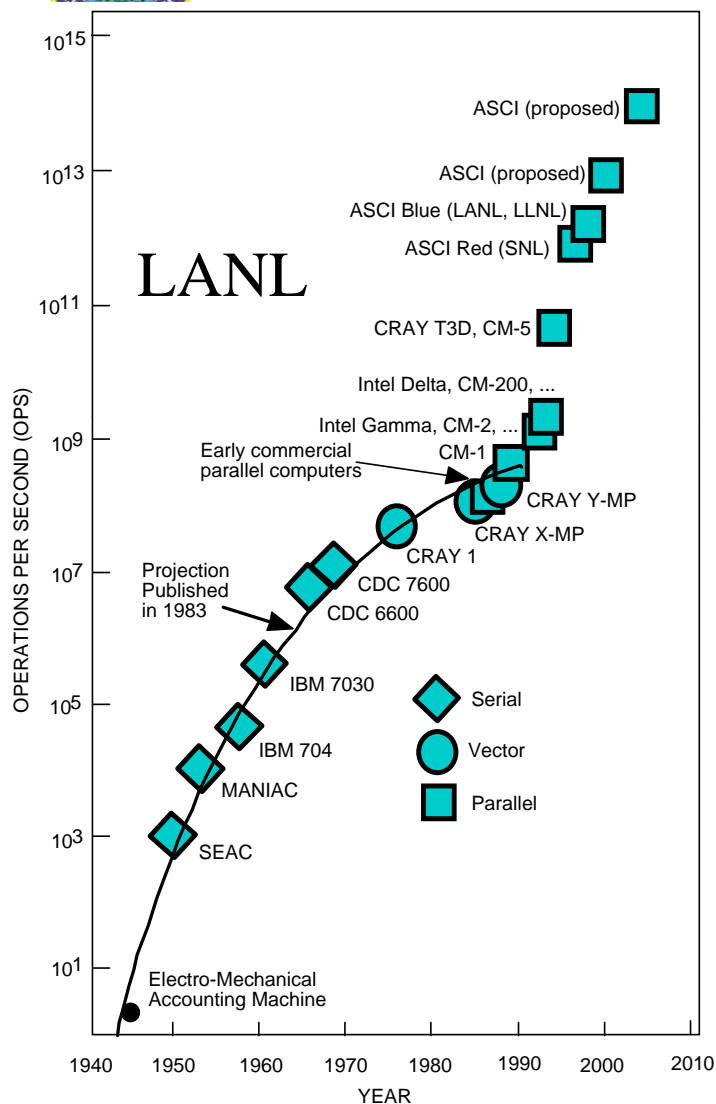
- | MSRC Systems | | | | | No. of Avail PEs | Peak GFLOPS of Actual PEs | Equiv. of 1,024-PE HABU |
|--------------|--|----------------------|---------------------------|-------------|------------------|---------------------------|-------------------------|
| | | Number of Actual PEs | HABU Rating per 1,024 PEs | Memory (GB) | | | |
| ERDC | SGI Origin 3900 | 1,024 | 3.08 | 1,024 | 1,008 | 1,434 | 3.08 |
| | Cray XT3 (Upgrade) | 8,320 | 11.54 | 16,640 | 8,192 | 43,264 | 93.76 |
| | Cray Hood | 8,848 | 10.39 | 17,696 | 8,608 | 40,701 | 89.76 |
| NAVO | IBM Regatta P4 | 2,944 | 6.55 | 5,968 | 2,832 | 20,019 | 18.83 |
| | IBM Cluster 1600 P5 | 2,976 | 12.31 | 5,952 | 2,816 | 20,237 | 35.78 |
| | IBM Cluster 1600 P5 | 1,504 | 13.66 | 3,008 | 1,408 | 10,227 | 20.06 |
| | IBM Regatta P4 | 1,408 | 2.10 | 1,408 | 1,328 | 7,322 | 2.89 |
| | IBM Regatta P4 | 512 | 6.55 | 736 | 464 | 3,482 | 3.28 |
| ARL | SGI Altix Cluster (D) | 256 | 8.68 | 256 | 256 | 1,536 | 2.17 |
| | IBM Opteron Cluster | 2,372 | 4.73 | 3,456 | 2,304 | 10,437 | 10.96 |
| | Linux Networx Xeon Cluster | 2,100 | 5.80 | 4,096 | 2,048 | 12,852 | 11.89 |
| | Linux Networx Woodcrest Cluster | 4,286 | 16.07 | 8,572 | 4,160 | 51,432 | 67.26 |
| | Linux Networx Dempsey Cluster | 3,360 | 10.86 | 6,720 | 3,336 | 21,504 | 35.63 |
| | Linux Networx Cluster | 256 | 5.21 | 256 | 256 | 1,567 | 1.30 |
| ASC | IBM Regatta P4 (D) | 32 | 2.55 | 32 | 32 | 166 | 0.08 |
| | SGI Origin 3900 | 2,048 | 3.08 | 2,048 | 2,032 | 2,867 | 6.16 |
| | SGI Origin 3900 (D) | 128 | 1.90 | 128 | 128 | 179 | 0.24 |
| | HP Opteron Cluster | 2,048 | 6.71 | 4,096 | 2,048 | 10,650 | 13.42 |
| | SGI Altix Cluster | 2,048 | 6.84 | 2,048 | 2,000 | 12,288 | 13.68 |
| | SGI Altix 4700 (Density) | 256 | 12.02 | 1,024 | 250 | 1,638 | 3.00 |
| | SGI Altix 4700 (8192 2GB Density, 1024 4GB Memory) | 9,216 | 12.02 | 22,528 | 9,000 | 58,982 | 108.14 |
| MSRC Totals | | | | | 54,506 | 332,784 | 541.4 |

10/6/2009





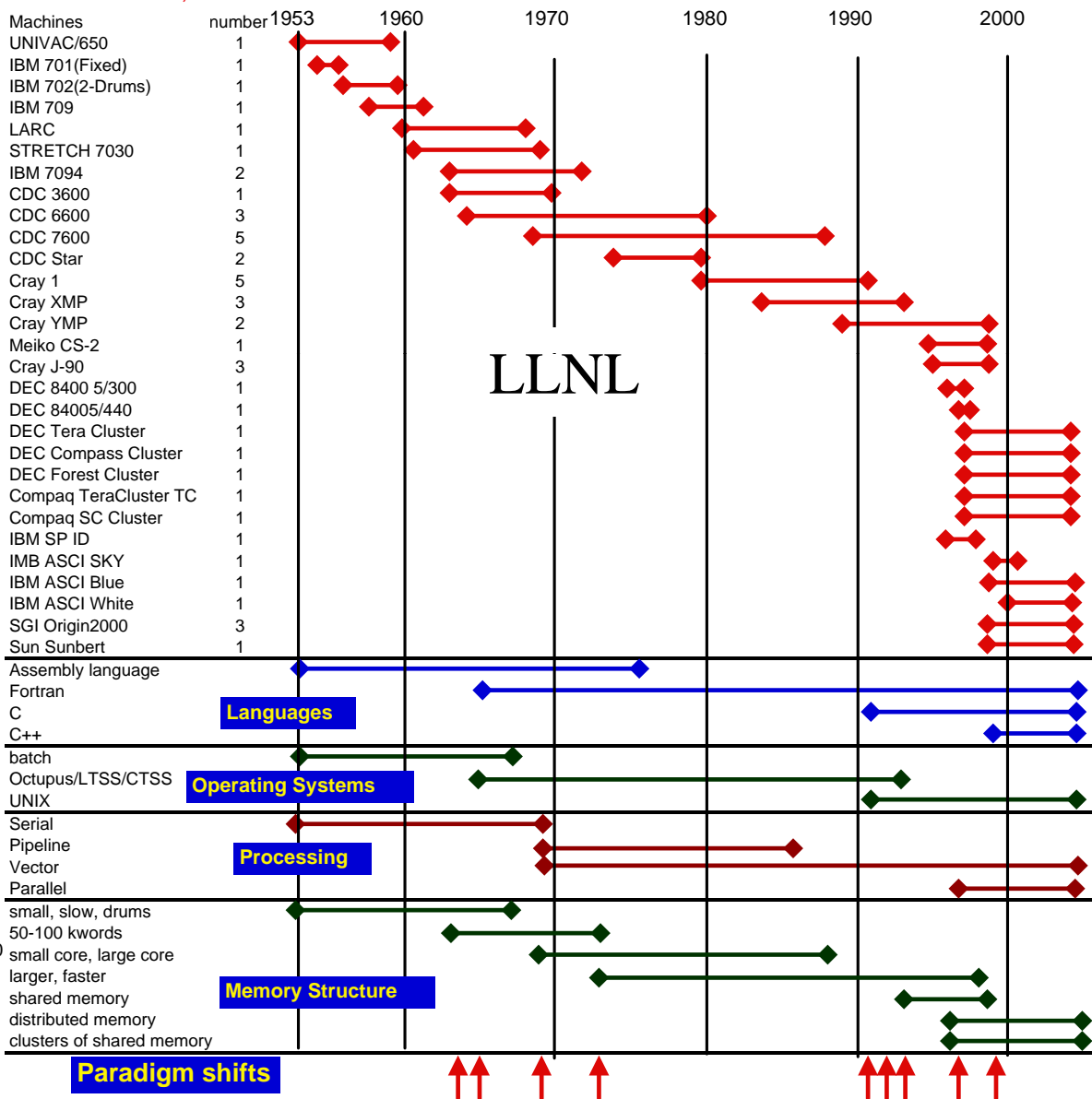
LLNL and LANL have had a new supercomputers roughly every 3 years since 1943 & a new programming paradigm every 10-15 years



—James Mercer-Smith

Post and Cook, 2000

Joe Requa, Doug Post





How do we get to the Brave New World? Brand new codes or improvements of existing codes?

❖ **Developing new codes is challenging!**

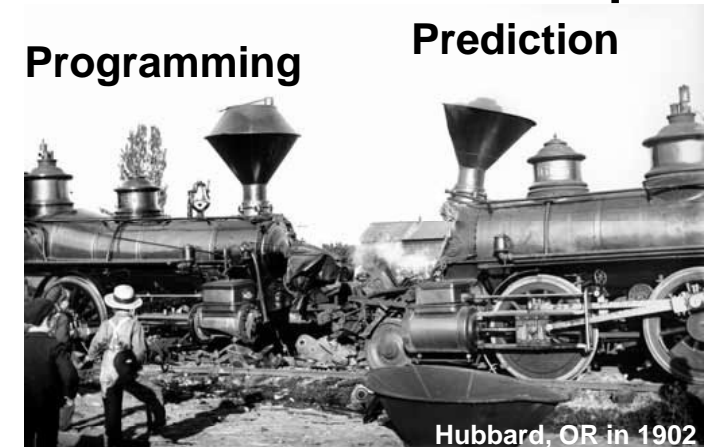
- Requires large (10 to 30 professionals), multi-disciplinary, multi-institutional teams
 - Takes 5 to 10 years
 - Requires extensive verification and validation
 - Requires a transition path to the user community
 - How many people would use Windows if almost everyone else used Mac OS or LINUX or UNIX...?
- ❖ For engineering codes, the practical approach is to port/upgrade existing tools and develop new ones where necessary
- ❖ There's no practical alternative to porting
- Independent software vendors are porting very slowly
 - “Reuse” is essential, a different use of “reuse”
 - Reuse the code, not individual components in other apps



Three Challenges

Performance, Programming and Prediction

1. **Performance Challenge** - Computers power increasing through growing complexity
 - Massive parallelization, multi-core & heterogeneous (CELL, FPGA, GPU...) processors, complex memory hierarchies.....
 2. **Programming Challenge** -Programming for Complex Computers
 - Rapid code development of codes with good performance
 3. **Prediction Challenge** —Developing **predictive** codes with complex scientific models
 - Develop accurate predictive codes
 - Verification
 - Validation
 - Code Project Management
- ❖ Train wreck coming between the last two



- ❖ **Better software development and production tools are desperately needed for us to take full advantage of computers**



Computational Engineering Requires a Village!

We need a complete problem solving capability:



Computers



Codes



V&V



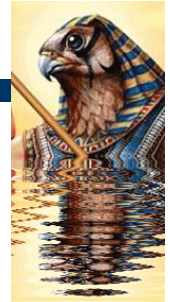
Users



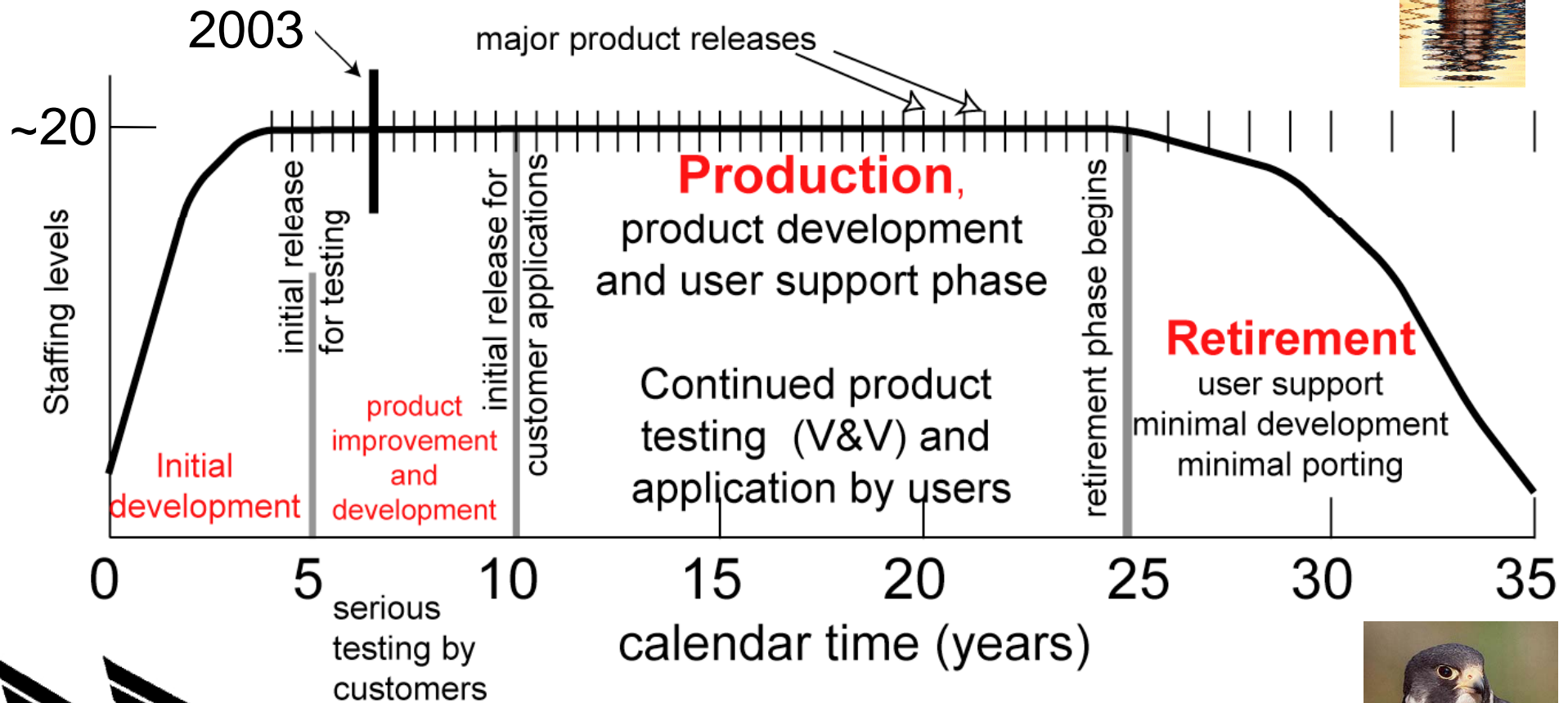
Sponsors



Developing a Large, Multi-scale, Multi-effect Code Takes a Large Team a Long Time



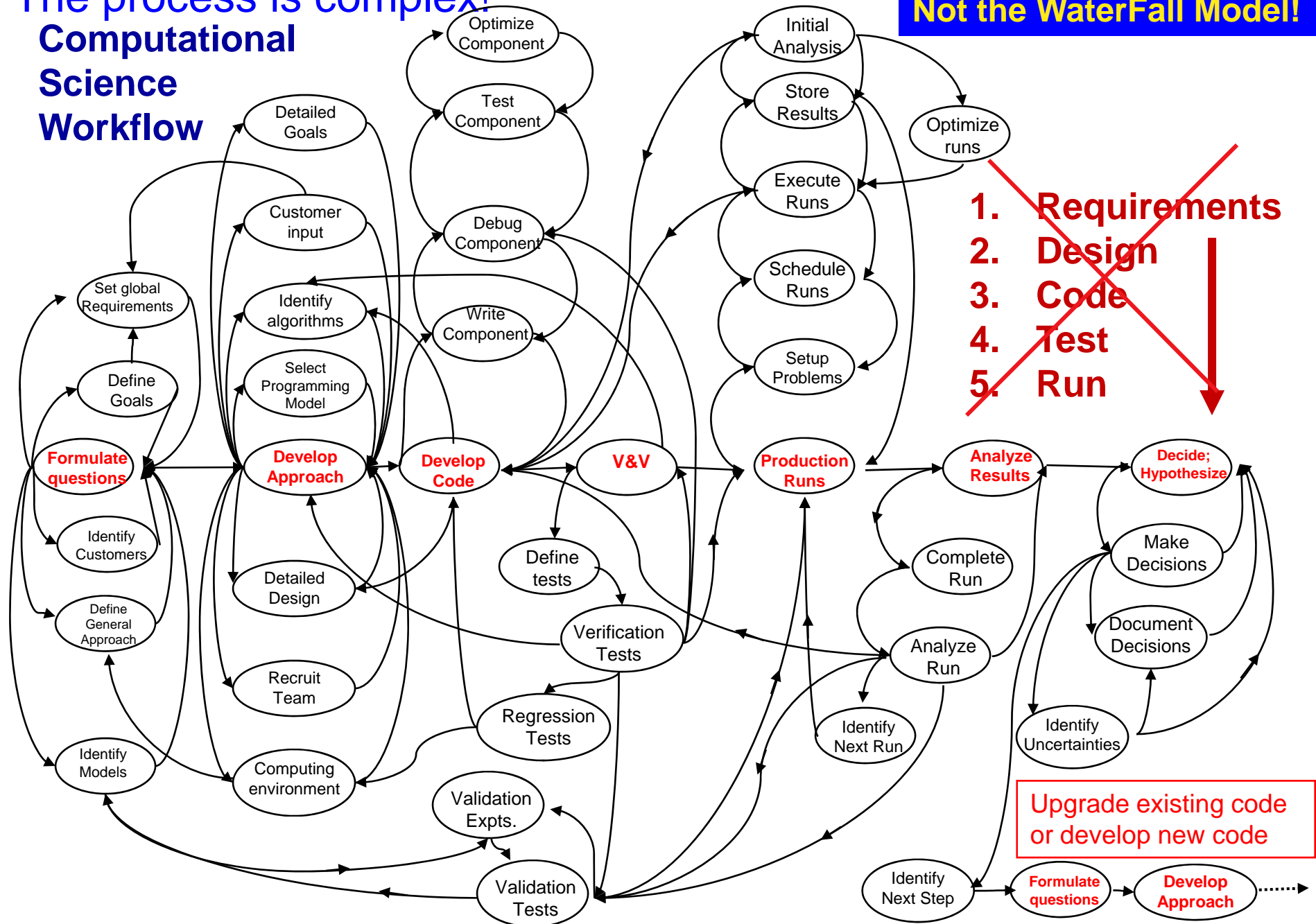
Falcon Project Life Cycle



The process is complex!

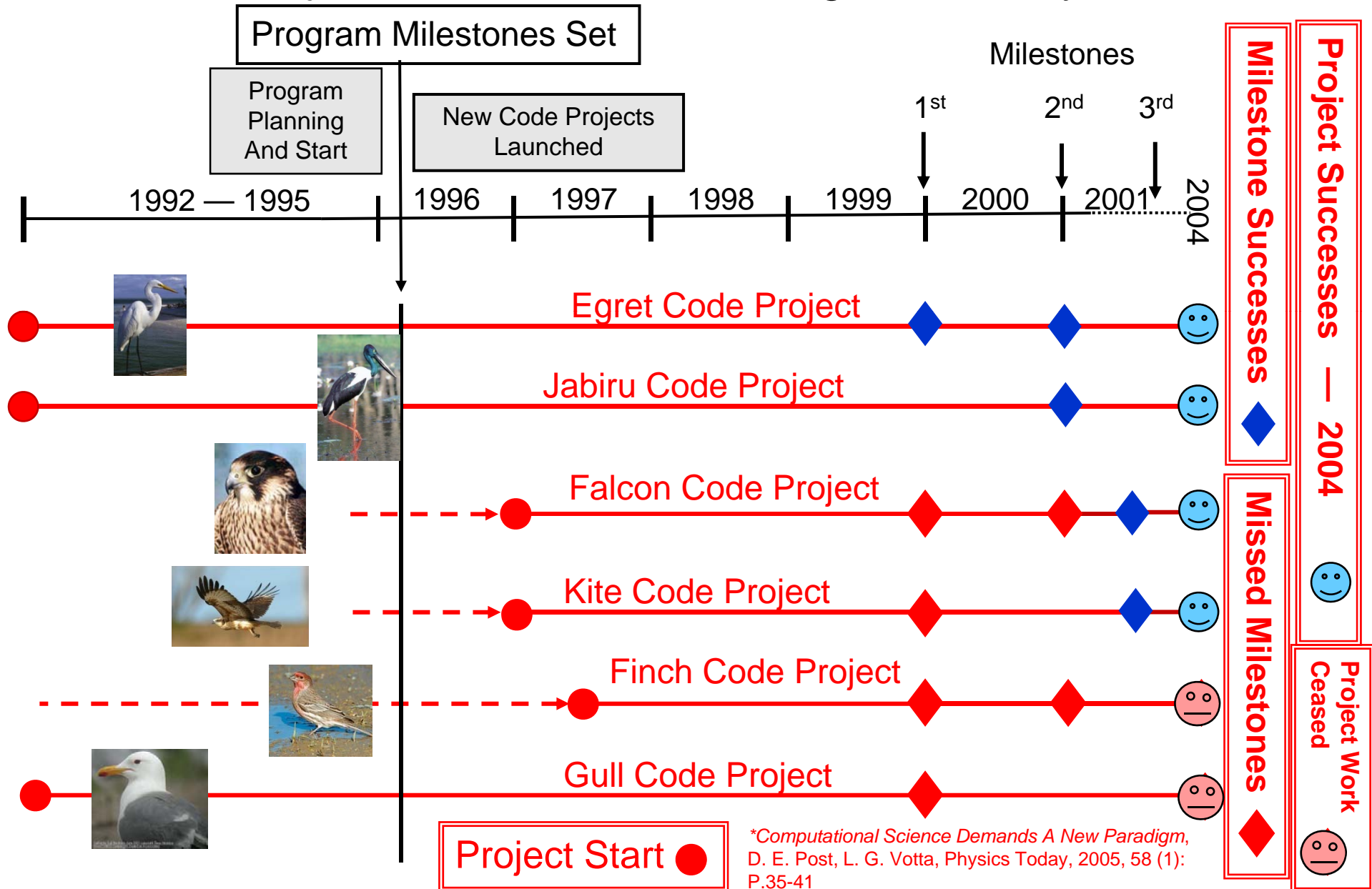
Computational Science Workflow

Not the WaterFall Model!



The Process has large risks!*

Code Project Schedule For Six Large-scale Physics Codes



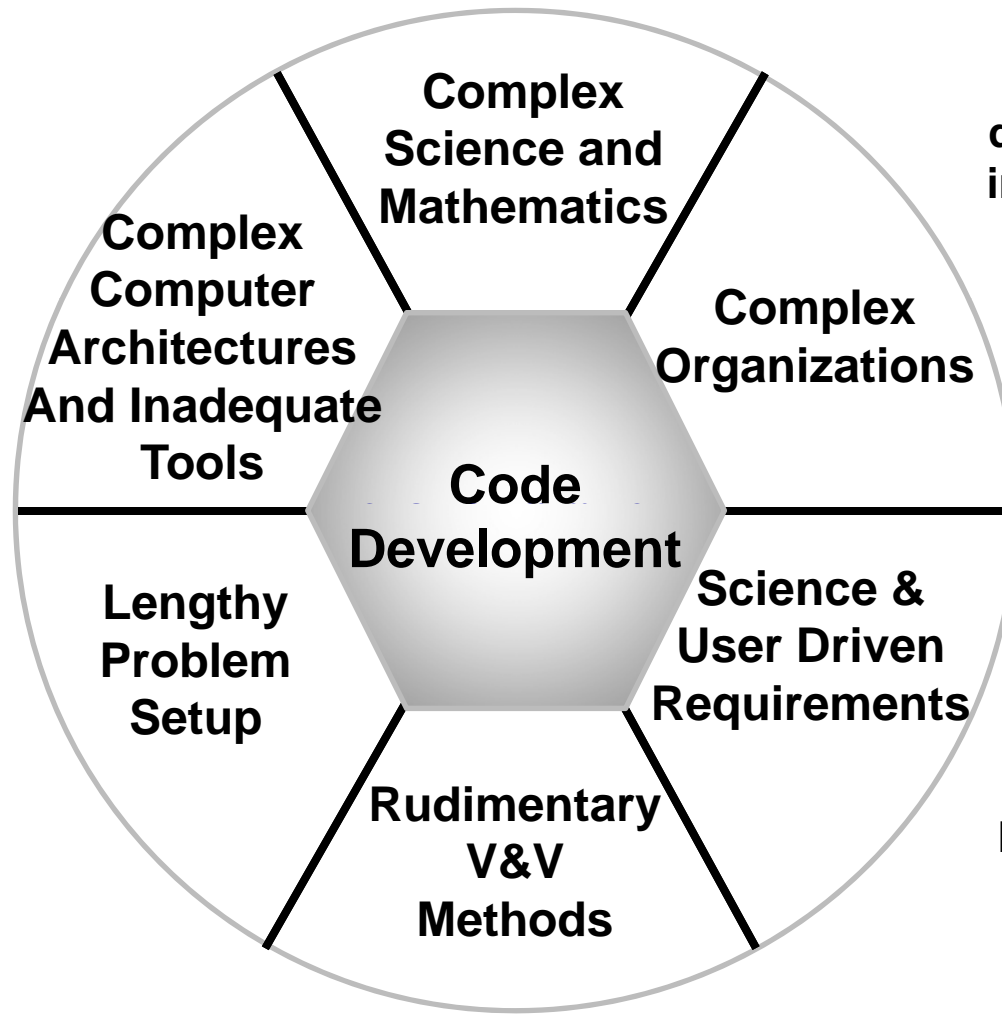


Computational Engineering Code Developer's World – Six Major Challenges and Risks

Many strongly coupled effects and massively parallel computers

Zillions of complex processors linked with complicated and slow networks + Little help for dealing with this complexity

Large, multi-disciplinary, multi-institutional teams



Problem setup (e.g. mesh generation) takes too long for rapid design development

Laws of nature & user needs win every time

Immature methods and few validation experiments



What are the characteristics of the DoD Big Codes (and DOE big codes)

- ❖ Surveyed DoD codes to verify characterizations of CSE codes.
- ❖ Identify general characteristics

Questionnaire asked for:

- ❖ Contact information
- ❖ Code purpose
- ❖ Team size, number of users
- ❖ Domain Science area and sponsor
- ❖ Code size (slocs)
 - Total and for each language
- ❖ Code history
 - How long did the code take to develop and how old is it now?)
- ❖ Platforms
- ❖ Degree of parallelism
- ❖ Computer time usage
- ❖ Memory requirements
- ❖ Algorithms



Surveyed the top 40 DoD codes (ordered by time requested), 15 responses.

Application Code	Hours		Application Code	Hours	
CTH	93,435,421		DMOL	5,200,100	
HYCOM	89,005,100		ICEM	4,950,000	
GAUSSIAN	49,256,850		CFD++	5,719,000	
ALLEGRA	32,815,000		ADCIRC	4,100,750	
ICEPIC	26,500,000		MATLAB	4,578,430	
CAML	21,000,000		NCOM	5,080,000	
ANSYS	17,898,520		Loci-Chem	5,500,000	
VASP	18,437,500		GAMSS	5,142,250	
Xflow	15,165,000		STRIPE	4,700,000	
ZAPOTEC	12,125,857		USM3D	4,210,000	
XPATCH	23,462,500		FLUENT	3,955,610	
MUVES	10,974,120		GASP	4,691,000	
MOM	18,540,000		Our DNS code (DNSBLB)	2,420,000	
OVERFLOW	8,835,500		ParaDis	4,000,000	
COBALT	14,165,750		FLAPW	4,050,000	
Various	8,125,000		AMBER	4,466,000	
ETA	11,700,000		POP	3,800,000	
CPMD	5,975,000		MS-GC	3,500,000	
ALE3D	5,864,500		TURBO	3,600,600	
PRONTO	5,169,100		Freericks Solver	2,600,000	



Characteristics Aren't Surprising.

	Team size FTEs	# users	Total sloc(k)	SLOC Fortran 77 (k)	SLOC Fortran 90, 95 (k)	SLOC C (k)	SLOC C++ (k)	other
Mean	38	5,038	820	24%	34%	17%	13%	13%
Median	6	27	275					

- ❖ Even now, codes are developed by teams
- ❖ Most codes have more users than just the development team
- ❖ Codes are big
- ❖ 58% of the codes are written in Fortran.
- ❖ New languages with higher levels of abstraction are attractive, but they will have to be compatible and interoperable with Fortran with MPI.



Further Data Isn't Surprising Either.

	Total project age	age production version	total number of different platforms	Largest Degree of Parallelism	Typical minimum # of processors	Typical Maximum # of processors	Is memory a limitation?	Memory processor GBytes /proc
Mean	19.8	15.1	6.9	1000 to 3000	225	292	Sometimes	0.75-4
Median	17.5	15.5	7.0	1000 to 3000	128	128		

- Most codes are at least 15 years old
- Most codes run on at least 7 different platforms
- Most codes can run on ~1000 processors, but don't
- Most users want at least 1 GByte / processor of memory.



5 detailed case studies of CSE codes make similar observations.

	Falcon	Hawk	Condor	Eagle
Application Domain	Product Performance	Manufacturing	Product Performance	Signal Processing
Project Duration	~10 years (since 1995)	~6 years (since 1999)	~20 years (since 1985)	~3 years
Number of Releases	9 Production	1	7	1
Earliest Predecessor	1970s	early 1990s	1969	?
Staffing	15 FTEs	3 FTEs	3-5 FTEs	3FTEs
Customers	<50	10s	100s	Demonstration code
Nonimal Code Size	~405,000	~134,000	~200,000	<100,000
Primary Languages	F77 (24%), C (12%)	C++ (67%), C (18%)	Fortran 77 (85%)	C++, Matlab
Other Languages	F90,Python,Perl,ksh/ csh/sh	Python, Fortran 90	Fortran 90, C, Slang	Java Libraries(~70%)
Target Hardware	Parallel Supecomputers	Parallel Supercomputers	PCs to Parallel Supercomputers	Embedded App
Status	Production	Production ready	Production	Demonstration code
Sponsors	DOE	DoD	DoD	DoD





Software Development Tools were identified.

	Falcon	Hawk	Condor	Eagle	Nene
Code Development Environment					
Compilers	F77, F90, C	C++,C, Fortran,Java	F77, F90	C++, Matlab,Java	F77,C
Scripts	Perl,Python,ksh,csh,sh,SCHEME,Gmake	Python	None	csh,perl,make,cmake,ANT	C Shell
Debuggers	TotalView, SourceForge	Valgrind, gbd	TotalView, gbd	TotalView, gbd, DBX	print+FTNCHK
Performance Monitoring	Pixie,DCPI,Speedshop, Prof	Speedshop, PAPI	None	Mercury TATL	NetPIPE
Domain Decomposition		Metis			
Execution Environment					
Element Generation		CAD ProE	In-house tools	N/A	Data basis sets
Visualization		ICE,VTK, Paraview, Tecplot	CEI Ensignt, Paraview	Matlab	Local product
Data Analysis		XDMF (supports Paraview)		Matlab	Local Product
Code Development Process Tools					
Configuration Management	CVS	CVS	CVS	Perforce, Subversion	Manual
Bug Tracking		Custon(~Bugzilla)	None	no formal system	no formal system
Code Documentation	Web-based	Doxygen	MS Word	In-code comments	User documentation; in-code comments
Support Libraries					
Computational Mathematics		PETc, VSS,PSPASES,CG	In-House tools	FFTs	BLAS
Parallel Programming Libraries	MPI	MPI	MPI	MPI, PVL (~POOMA)	MPI, TCP/IP



Many Barriers and Challenges

- ❖ **V&V**
- ❖ **Changing computer architectures**
- ❖ **Parallel Scaling and Parallel Programming Models**
- ❖ **Complexity of Domain Science (strongly-coupled Multi-physics, multi-scale...)**
- ❖ **Cautious user community**
 - **Answers, not performance is not their ultimate goal**
 - **The better is the enemy of the good!**



Recommendations

- ❖ **Porting will be essential in the future**
- ❖ **Not just for HPC but for all computer programs venturing into tomorrow's multi-core heterogeneous world**
- ❖ **Tools should facilitate porting, most useful tools:**
 - **Reduce complexity**
 - **Hide complexity of computer in portable libraries**
 - **Simplify Verification**
 - **Preserve ability to link to many languages**
- ❖ **Also useful to improve Software Engineering**
 - **Documentation, modularity, interface standards, interoperability, scalability**



Software reuse in acquisition systems

- ❖ **Contractor-initiated reuse**
 - **Reuse from prior developments or investments**
 - **Product line approach to developing components – Fighter A/C radar for example**
 - **Reuse from outside sources**
 - **Signal processing libraries (VSIPL)**
 - **Purchased environments**
 - » **OS, Middleware, graphics toolkits**
 - **Open Source**



Software reuse in acquisition systems (cont'd)

- ❖ **Government-sponsored reuse**
 - **GFI from previous developments**
 - **Made available by Gov't Purpose Rights**
 - **Contracted collaborative environments**
 - **Central code repository specific to an acquisition or domain**
 - **Controlled by central CM agent**
 - **GFI from purchase**
 - **Government purchased license made available to offerors and developers**
 - » **Tactical Component Network**



Issues

❖ Assurance

- Is the reused code free from malware and vulnerabilities?

❖ Performance

- How do we know what the code really does?
- How does its use affect system properties?
- Who is responsible? How much testing?

❖ Efficiency

- Realized % reuse from previous developments rarely meets initial estimates!!

❖ Intellectual property

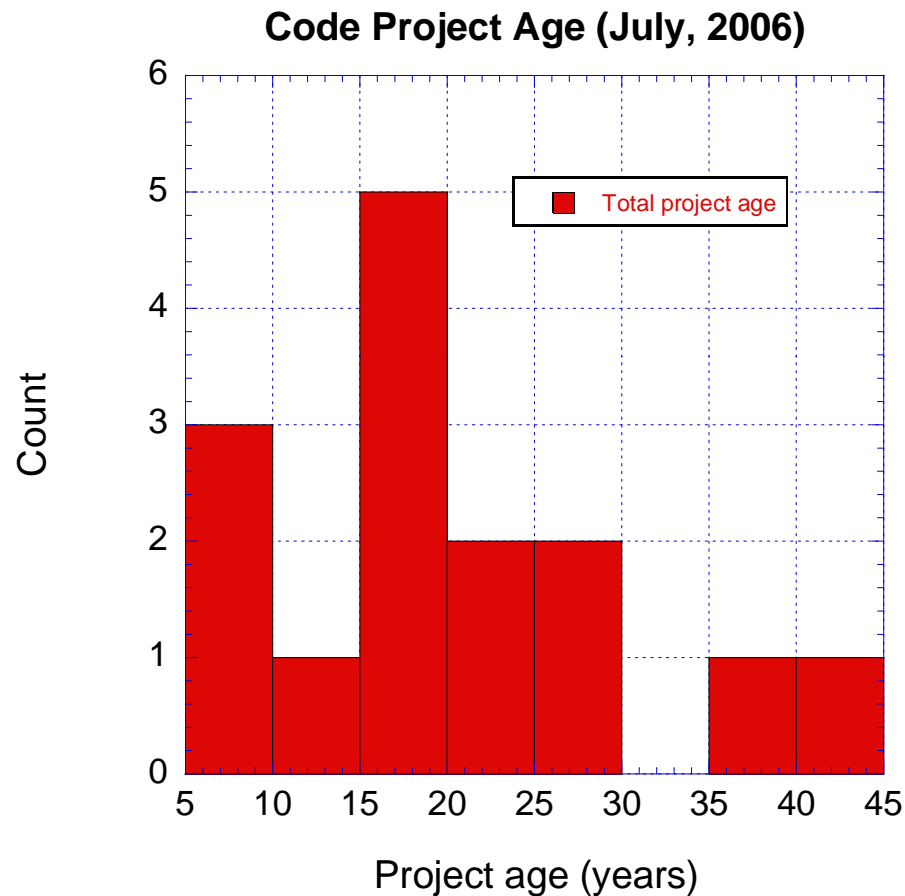
- Can we reuse the patented technologies in the code?
- Can we derive other works?



❖ Back-up Slides



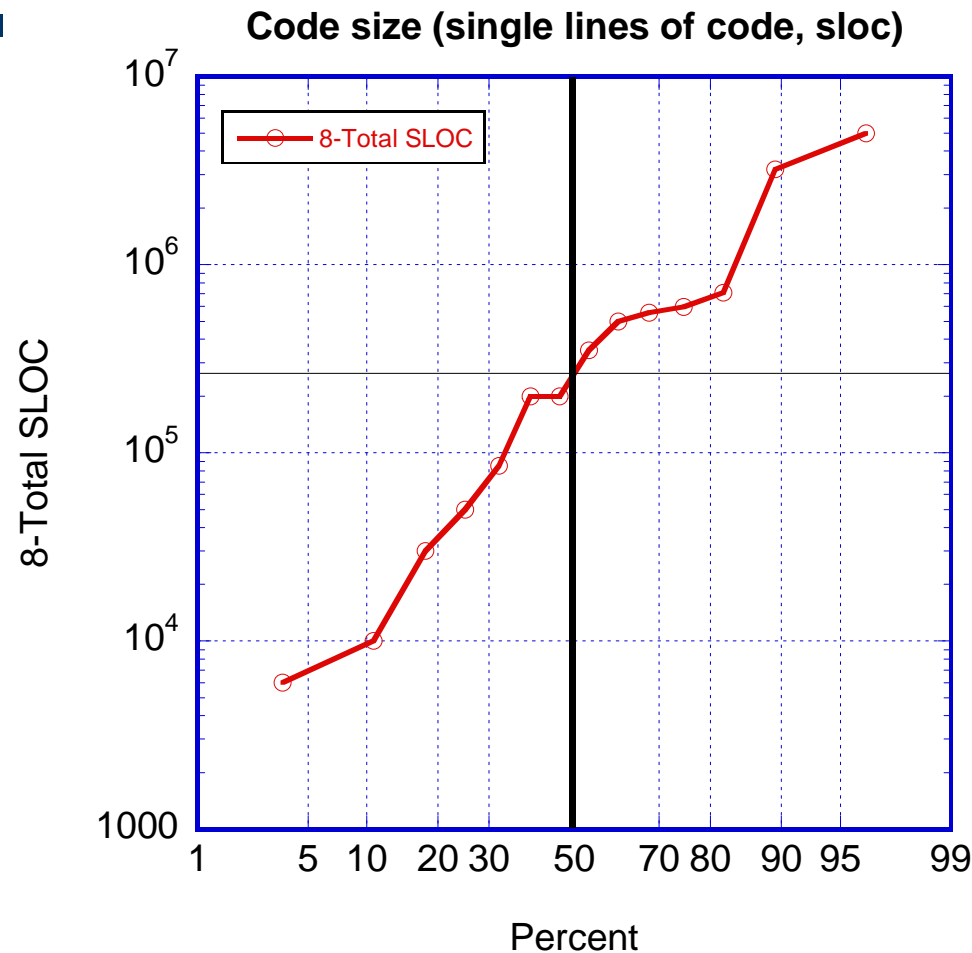
Most projects are at least 15 years old (and had predecessors).



- Almost all the codes that will run on platforms delivered within the next 5 years **exist now**.



Median code size is ~ 300,000 slocs.

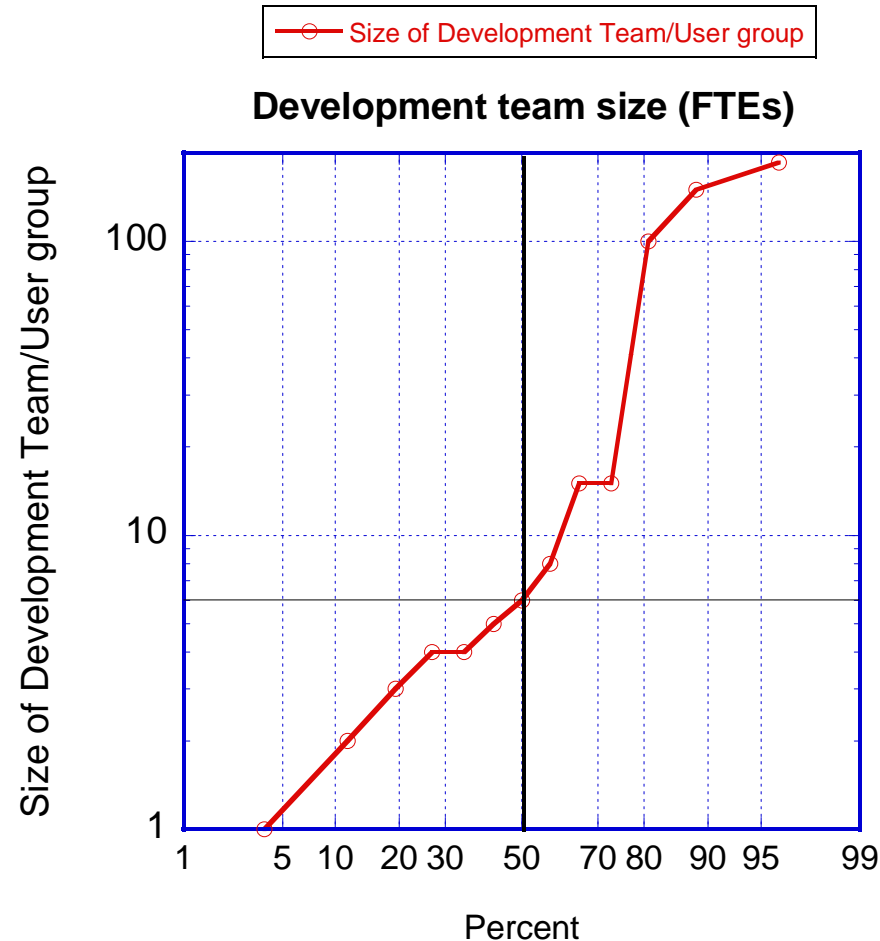


- Most codes will take 5 years or more to develop¹.

¹D. E. Post and R. P. Kendall, *International Journal of High Performance Computing Applications*, 18(2004), pp. 399-416



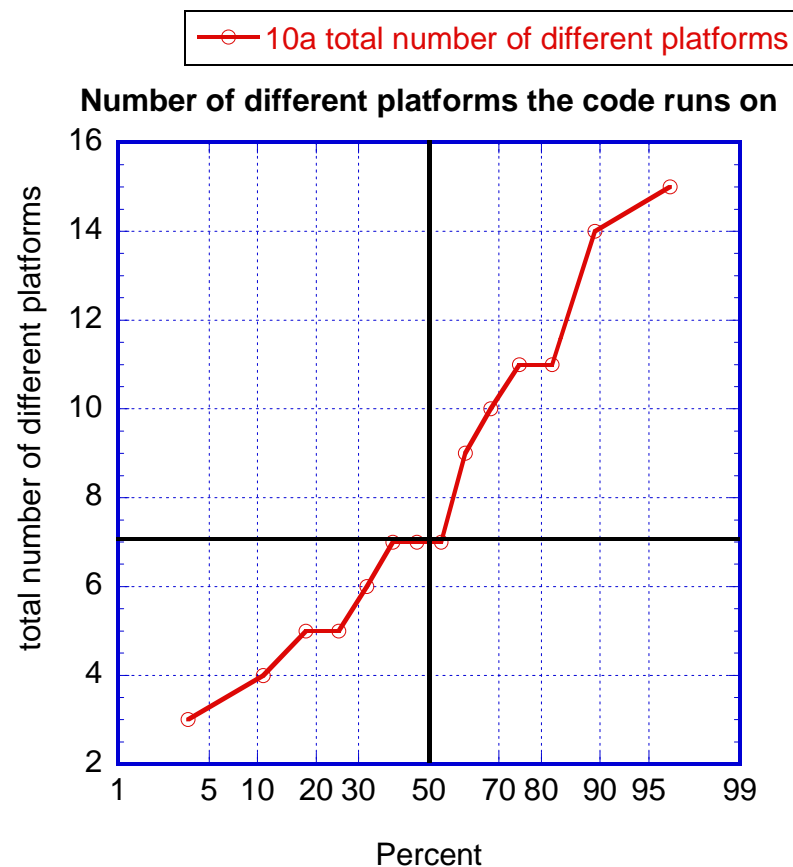
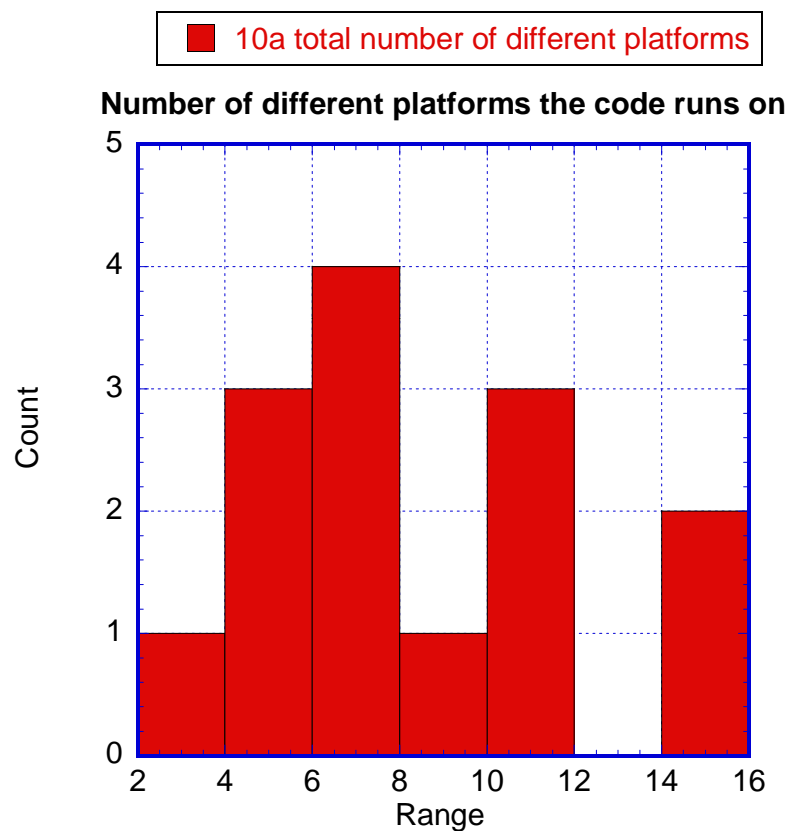
Median team size is 6 FTEs.



- Teamwork will be essential for new codes, especially for petaflop computing.



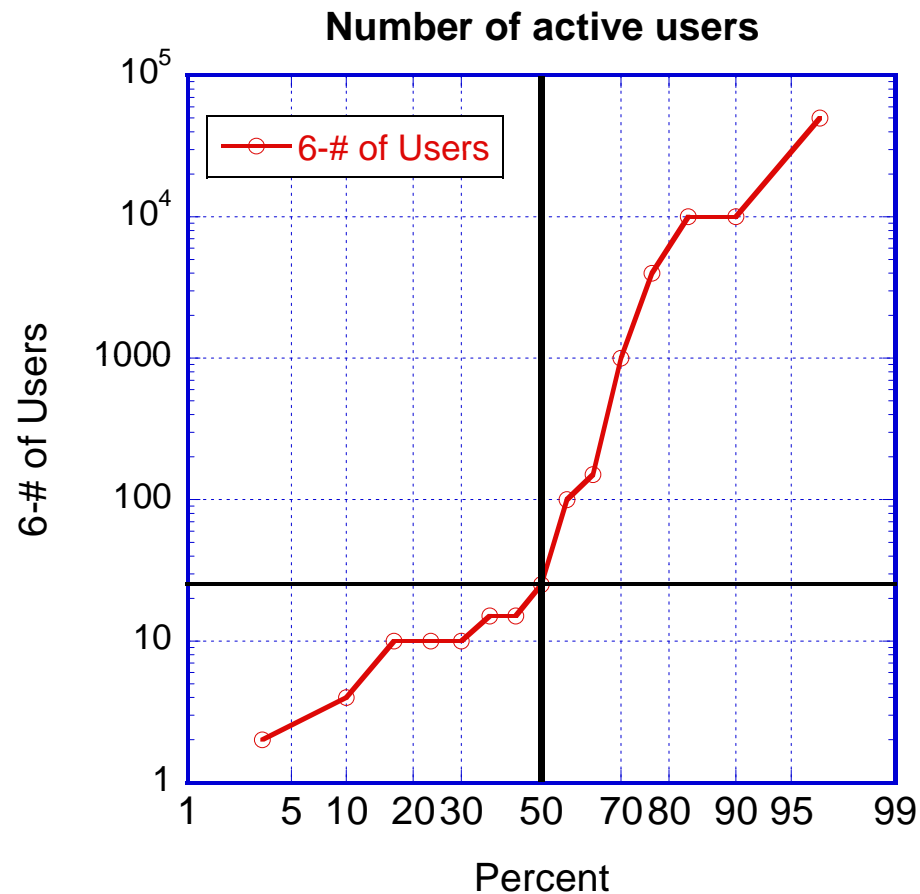
Median code runs on 7 different platforms.



- Code portability is a key, if not dominant, priority for code developers.



Median code has ~ 25 users.

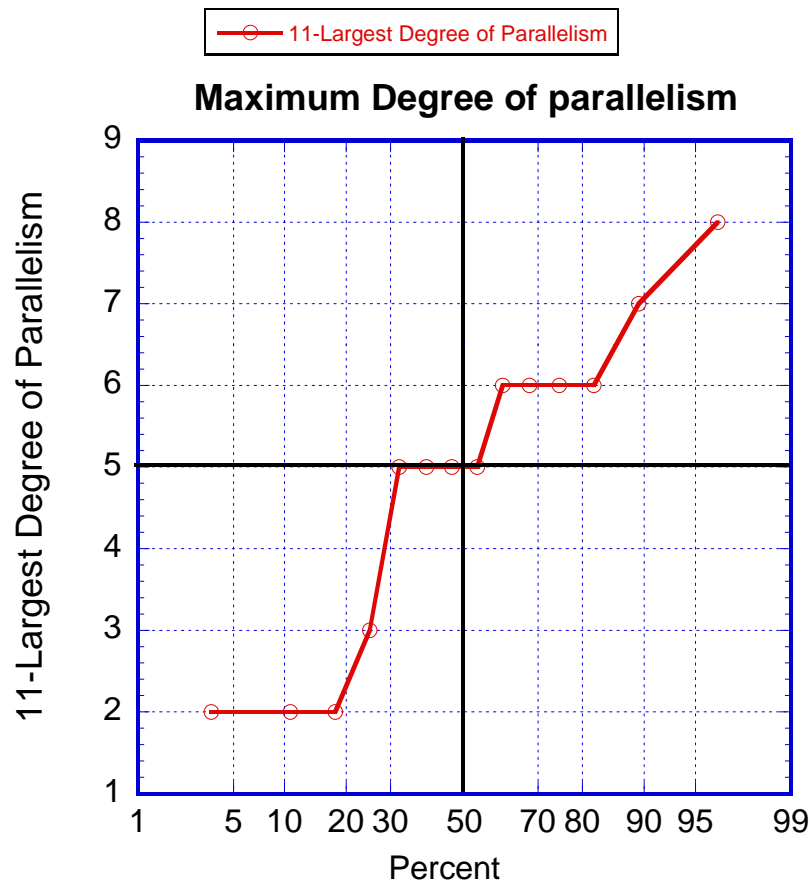


- User support and acceptance will be essential for success
- Support for code maintenance will be essential!



Median code is fairly parallel.

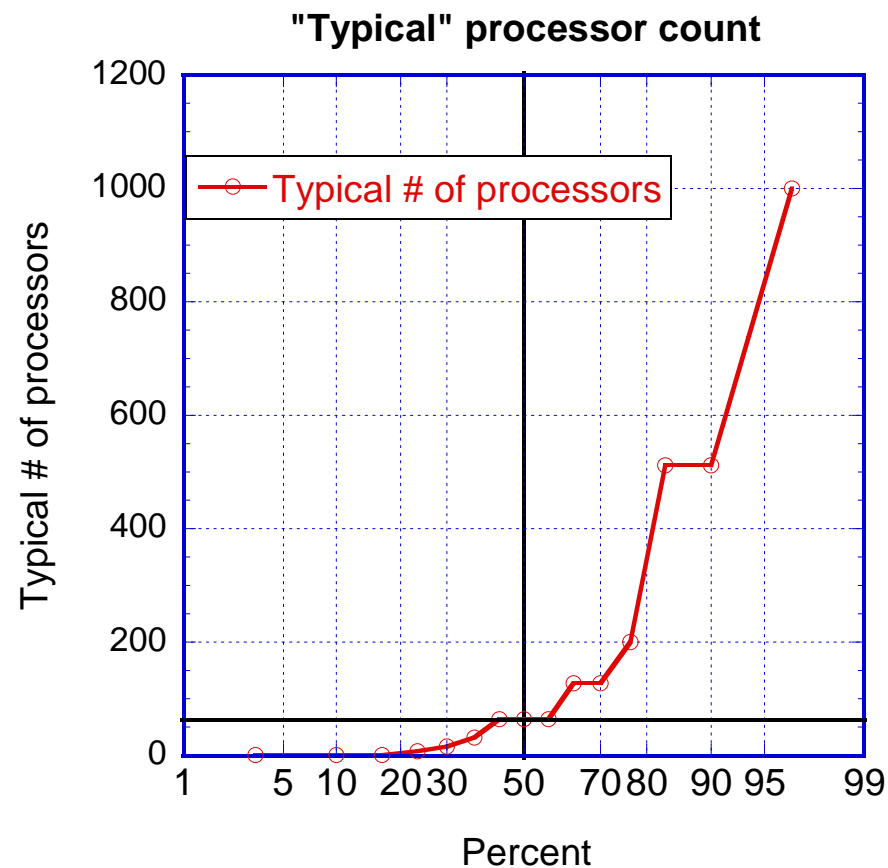
8. > 30,000 processors
7. 10,001 to 30,000 processors
6. 3,001 to 10,000 processors
5. 1,001 to 3000 processors
4. 300 to 1,000 processors
3. 101 to 300 processors
2. 11 to 100 processors
1. Less than 10 processors



- We have to scale from 100-3,000 processors to 50,000-200,000 processors in two years to achieve petaflop performance.



“Routine” processor count is much less than peak.



- We have to scale from 30-200 processors to 20,000-200,000 processors in two years to achieve petaflop performance.



58% of the codes are predominantly written in Fortran.

	Team size FTEs	# users	Total sloc(k)	SLOC Fortran 77 (k)	SLOC Fortran 90, 95 (k)	SLOC C (k)	SLOC C++ (k)	other
Mean	38	5,038	820	24%	34%	17%	13%	13%
Median	6	27	275					

❖ New languages with higher levels of abstraction are attractive, but they will have to be compatible and inter-operable with Fortran with MPI.



Most runs don't use a lot of processors.

	Total project age	age production version	total number of different platforms	Largest Degree of Parallelism	Typical minimum # of processors	Typical Maximum # of processors	Is memory a limitation?	Memory processor GBytes /proc
Mean	19.8	15.1	6.9	1000 to 3000	225	292	Sometimes	0.75-4
Median	17.5	15.5	7.0	1000 to 3000	128	128		

- Most users want at least 1 GByte / processor of memory.



HPCMP TI-05 Application Benchmark Codes perform differently on different platforms.

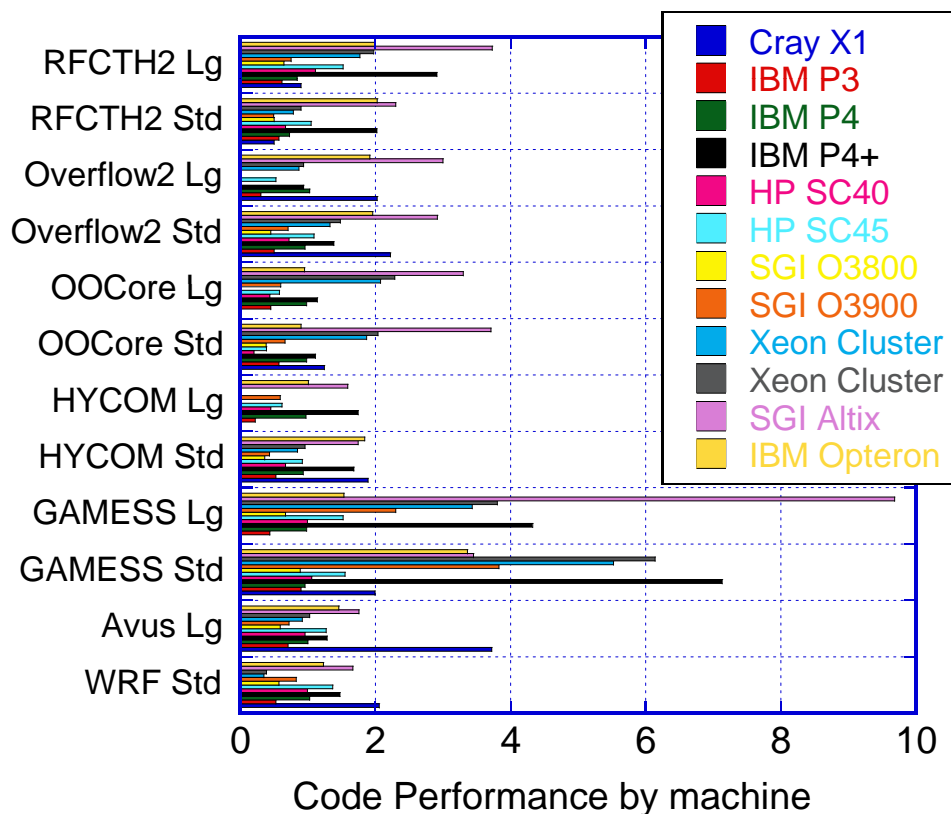
- ❖ **Aero – Aeroelasticity CFD code
(Fortran, serial vector, 15,000 lines of code)**
- ❖ **AVUS (Cobalt-60) – Turbulent flow CFD code
(Fortran, MPI, 19,000 lines of code)**
- ❖ **GAMESS – Quantum chemistry code
(Fortran, MPI, 330,000 lines of code)**
- ❖ **HYCOM – Ocean circulation modeling code
(Fortran, MPI, 31,000 lines of code)**
- ❖ **OOCore – Out-of-core solver
(Fortran, MPI, 39,000 lines of code)**
- ❖ **CTH – Shock physics code (SNL)
(~43% Fortran/~57% C, MPI, 436,000 lines of code)**
- ❖ **WRF – Multi-Agency mesoscale atmospheric modeling code
(Fortran and C, MPI, 100,000 lines of code)**
- ❖ **Overflow-2 – CFD code originally developed by NASA
(Fortran 90, MPI, 83,000 lines of code)**



Performance depends on the computer and on the code.

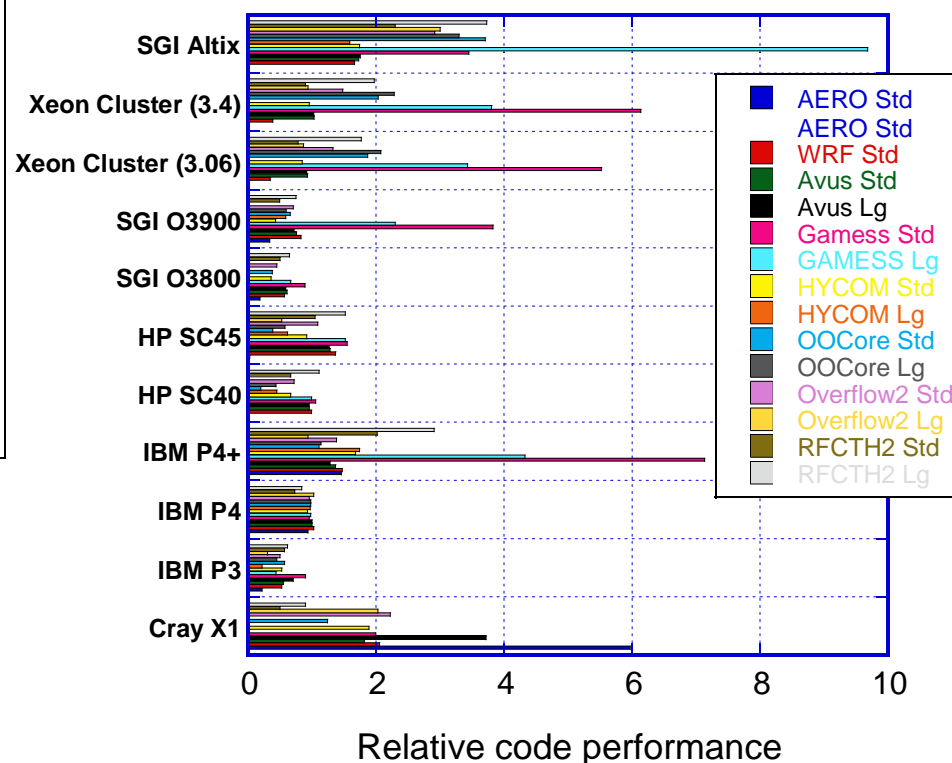
- Normalized Performance = 1 on the NAVO IBM SP3 (HABU) platform with 1024 processors (375 MHz Power3 CPUs) assuming that each system has 1024 processors.
- GAMESS had the most variation among platforms.

Code Performance (by machine)



Substantial variation of codes for a single computer.

Code performance (grouped by machine)



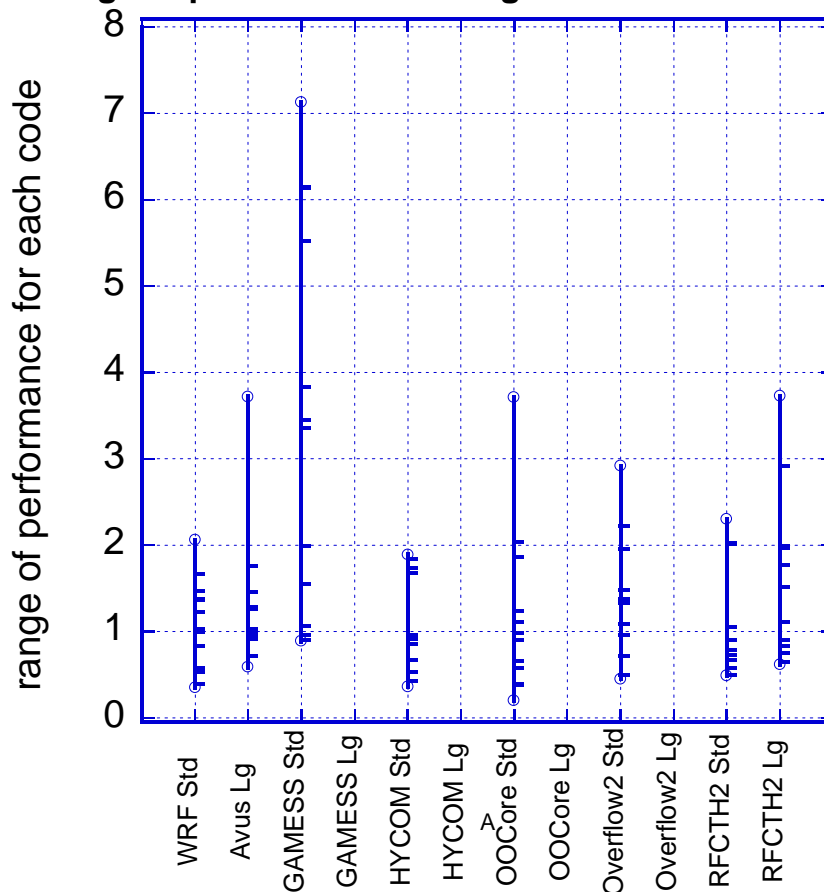
—SC 2005 panel Tour de HPCycles





Performance range of codes is large.

Range of performance among machines for each code





General conclusions

- ❖ **Performance depends on application and on the computer**
 - No computer works best for all applications
 - A suite of applications requires a suite of computer types
- ❖ **Tuning for a platform can pay off in a big way**
- ❖ **Shared memory is really good for some codes**



We made 9 observations based on detailed case studies.

- ❖ **We made 9 observations from the five detailed case studies (Falcon, Hawk, Condor, Eagle, Nene).**
 - **These observations and conclusions were consistent with our prior, less detailed case studies.**
- ❖ **These 9 observations help identify the issues to focus on for petaflop applications.**





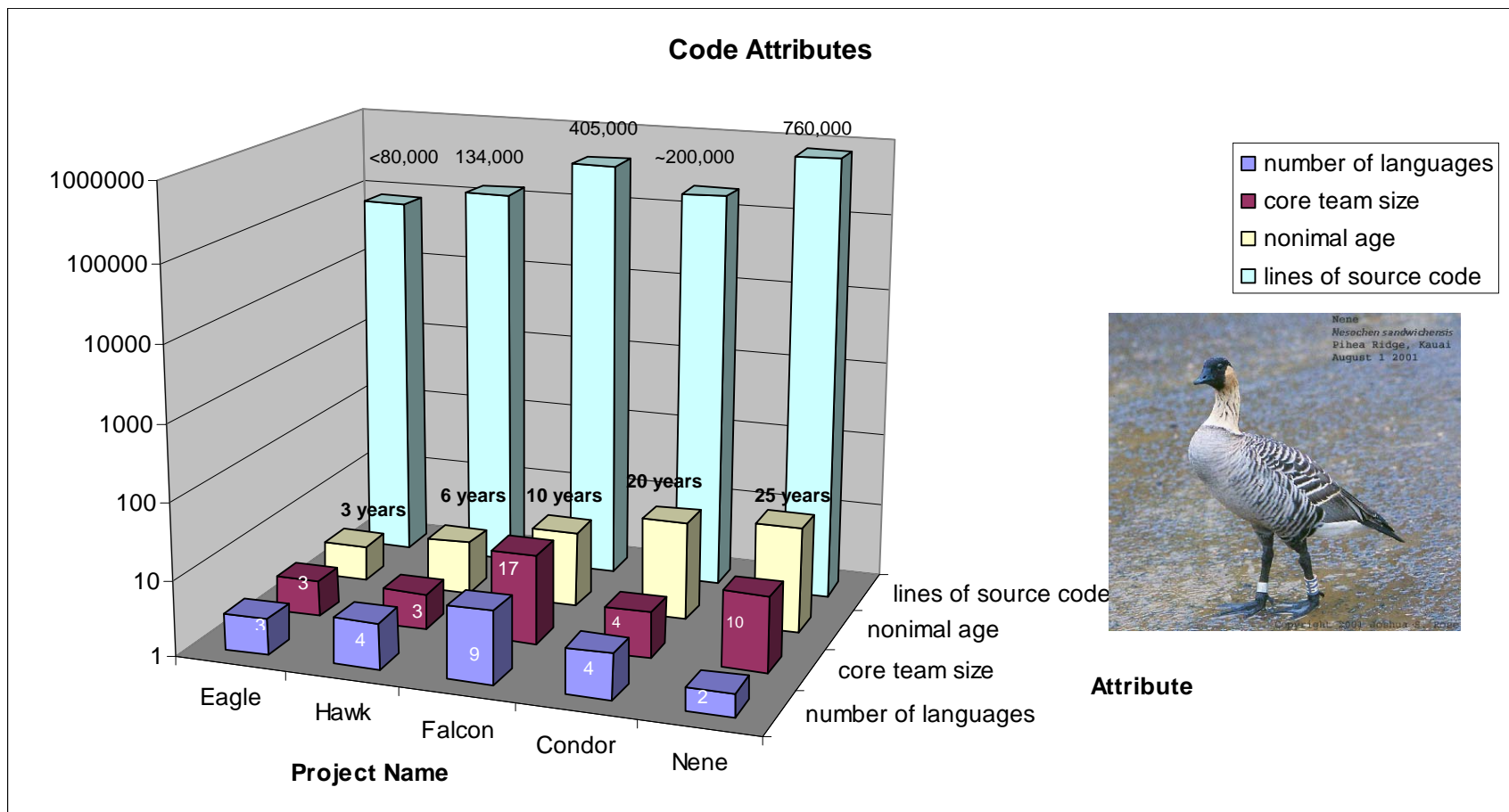
Nine Cross-Study Observations

1. Once selected, the primary languages (typically Fortran) adopted by existing code teams do not change.
2. The use of higher level languages (e.g. Matlab) has not been widely adopted by existing code teams except for "bread-boarding" or algorithm development.
3. Code developers in existing code teams like the flexibility of UNIX command line environments.
4. Third party (externally developed) software and software development tools are viewed as a major risk factor by existing code teams.
5. The project goal is scientific discovery or engineering design. "Speed to solution" and "execution time" are not highly ranked goals for our existing code teams unless they directly impact the science.
6. All but one of the existing code teams we have studied have adopted an "agile" development approach.
7. For the most part, the developers of existing codes are scientists and engineers, not computer scientists or professional programmers.
8. Most of the effort has been expended in the "implementation" workflow step.
9. The success of all of the existing codes we have studied has depended most on keeping their customers (not always their sponsors) happy.





Summary of Code Attributes





Codes primarily use one or two programming languages, but utilize many others for special purposes.

	Falcon	Hawk	Condor	Eagle	Nene
Application Domain	Product Performance	Manufacturing	Product Performance	Signal Processing	Process Modeling
Project Duration	~10 years (since 1995)	~6 years (since 1999)	~20 years (since 1985)	~3 years	~25 years (since 1982)
Number of Releases	9 Production	1	7	1	?
Earliest Predecessor	1970s	early 1990s	1969	?	1977-78
Staffing	15 FTEs	3 FTEs	3-5 FTEs	3FTEs	~10FTEs+100s of contributors
Customers	<50	10s	100s	Demonstration code	~100,000
Nonimal Code Size	~405,000	~134,000	~200,000	<100,000	750,000
Primary Languages	F77 (24%), C (12%)	C++ (67%), C (18%)	Fortran 77 (85%)	C++, Matlab	Fortran 77 (95%)
Other Languages	F90,Python,Perl,ksh/c	Python, Fortran 90	Fortran 90, C, Slang	Java Libraries(~70%)	C (1%)
Target Hardware	Parallel Supecomputers	Parallel Supercomputers	PCs to Parallel Supercomputers	Embedded App	PCs to Parallel Supercomputers
Status	Production	Production ready	Production	Demonstration code	Production
Sponsors	DOE	DoD	DoD	DoD	DoD, DOE, NSF





We found sparse use of Software Metrics.

Metric	Falcon	Hawk	Condor	Eagle
Lines of code	x	x	x	x
Function points	x			
Stories, project velocity				
Cyclomatic complexity				
Data coupling				
Comment lines				
Locality				
Concurrency				
Defect rates				
Time-to-fix defects		x		
Number of debug runs/unit time				
Test Coverage	x	x	x	
Frequency that regression testing uncovers problems	x			
Code performance	x	x	x	x
Degree of performance optimization	x			x
Parallel scaling		x	x	x
Number of users		x	x	
Number of production runs/unit time				
Computer time for code development/unit time				
Computer time for production/unit time			x	





Cross-Study Observations

❖ Observation #1:

- Once selected, the primary languages (typically F77) adopted by existing code teams do not change.
 - Any new language will have to be compatible with existing languages and will, at best, be introduced only incrementally.
 - Migration to a new version of the language (e.g. F77 to F90) often occurs, but seldom to a different language (e.g. F77 to C)

Early users of a petaflop machine will require stable Fortran, C and C++ implementations with MPI libraries on the new hardware





Use of Higher-Level Languages

	Falcon	Hawk	Condor	Eagle	Nene
Application Domain	Product Performance	Manufacturing	Product Performance	Signal Processing	Process Modeling
Project Duration	~10 years (since 1995)	~6 years (since 1999)	~20 years (since 1985)	~3 years	~25 years (since 1982)
Number of Releases	9 Production	1	7	1	?
Earliest Predecessor	1970s	early 1990s	1969	?	1977-78
Staffing	15 FTEs	3 FTEs	3-5 FTEs	3FTEs	~10FTEs+100s of contributors
Customers	<50	10s	100s	Demonstration code	~100,000
Nonimal Code Size	~405,000	~134,000	~200,000	<100,000	750,000
Primary Languages	F77 (24%), C (12%)	C++ (67%), C (18%)	Fortran 77 (85%)	C++, Matlab	Fortran 77 (95%)
Other Languages	F90,Python,Perl,ksh/csh/sh	Python, Fortran 90	Fortran 90, C, Slang	Java Libraries(~70%)	C (1%)
Target Hardware	Parallel Supecomputers	Parallel Supercomputers	PCs to Parallel Supercomputers	Embedded App	PCs to Parallel Supercomputers
Status	Production	Production ready	Production	Demonstration code	Production
Sponsors	DOE	DoD	DoD	DoD	DoD, DOE, NSF





Cross-Study Observations

❖ Observation #2:

- The use of higher level languages (e.g. Matlab) has not been widely adopted by existing code teams.
- Higher level languages are utilized by some teams for “bread-boarding” and algorithm development followed by implementation in a lower level, but higher performance language

“I’d rather be closer to machine language than more abstract. I know even when I give very simple instructions to a compiler, it does not necessarily give me machine code that corresponds to that set of instructions...”

quote from Condor Technical Team Leader





Cross-Study Observations

❖ Observation #3:

- Code developers like the predictability, flexibility and universality of UNIX command line environments.

One of the reasons that IDE tools are not used is that “they try to impose a particular style of development on me and I am forced into a particular mold.”

quote from Eagle team leader

- Any new IDE will need to meet these requirements





Risk Aversion to the use of 3rd Party Tools

	Falcon	Hawk	Condor	Eagle	Nene
Code Development Environment					
Compilers	F77, F90, C	C++,C, Fortran,Java	F77, F90	C++, Matlab,Java	F77,C
Scripts	Perl,Python,ksh,csh,sh,SCHEME,Gmake	Python	None	csh,perl,make,cmake,ANT	C Shell
Debuggers	TotalView,SourceForge	Valgrind, gbd	TotalView, gbd	TotalView, gbd, DBX	print+FTNCHK
Performance Monitoring	Pixie,DCPI,Speedshop,Prof	Speedshop, PAPI	None	Mercury TATL	NetPIPE
Domain Decomposition		Metis			
Execution Environment					
Element Generation		CAD ProE	In-house tools	N/A	Gaussian basis sets
Visualization		ICE,VTK, Paraview, Tecplot	CEI Ensight, Paraview	Matlab	MACMOPLT
Data Analysis		XDMF (supports Paraview)		Matlab	MACMOPLT
Code Development Process Tools					
Configuration Management	CVS	CVS	CVS	Perforce, Subversion	Manual
Bug Tracking		Custon(~Bugzilla)	None	no formal system	no formal system
Code Documentation	Web-based	Doxygen	MS Word	In-code comments	User documentation; in-code comments
Support Libraries					
Computational Mathematics		PETc, VSS.PSPASES.CG	In-House tools	FFTs	BLAS
Parallel Programming Libraries	MPI	MPI	MPI	MPI, PVL (-POOMA)	MPI, TCP/IP





Performance Monitoring, Profile and Analysis Tools

- ❖ AIMS (NASA AIMS)
- ❖ **DCPI (DEC/Compaq/HP)**
- ❖ DEEP
- ❖ Dimenas (CEPBA Barcelona)
- ❖ Dynamic Probe Class Library (IBM)
- ❖ DynaProf (Univ. of Tennessee)
- ❖ FALCON (Georgia Tech)
- ❖ HPC Toolkit (Rice Univ.)
- ❖ HPM Toolkit (IBM)
- ❖ Jumpshot (Argonne-DOE)
- ❖ Monitor
- ❖ MPIMAP (LLNL)
- ❖ mpiP(ORNL/LLNL)
- ❖ Pable/SvPablo(Univ Illinois/Univ. North Carolina)
- ❖ PAPI Libraries (Univ. Tennessee)

- ❖ Paradyn (Univ. Wisconsin)
- ❖ Paraver ((CEPBA Barcelona)
- ❖ PDT (Univ. Oregon)
- ❖ PE Benchmark Toolset (IBM)
- ❖ Performance Toolkit (IBM)
- ❖ **prof/gprog/tprof/pgprof**
- ❖ Quantity (Rational/IBM)
- ❖ **Speedshop (SGI)**
- ❖ Tau (Univ. of Oregon)
- ❖ ThreadMon
- ❖ Timescan (Etnus)
- ❖ TRAPPER
- ❖ WARTS
- ❖ Vampir(Pallas, now Intel)
- ❖ Xprofiler (IBM)



Debugging/Visualization Tools

❖ Debugging

- **TotalView (Etnus)**
- Gdb (gnu)
- DXB
- Ladebug
- Great Circle (geodesic)

❖ Visualization

- **CEI Ensight**
- **Gnuplot**
- IDL
- Kaleidagraph
- **Paraview**





Cross-Study Observations

❖ Observation #4:

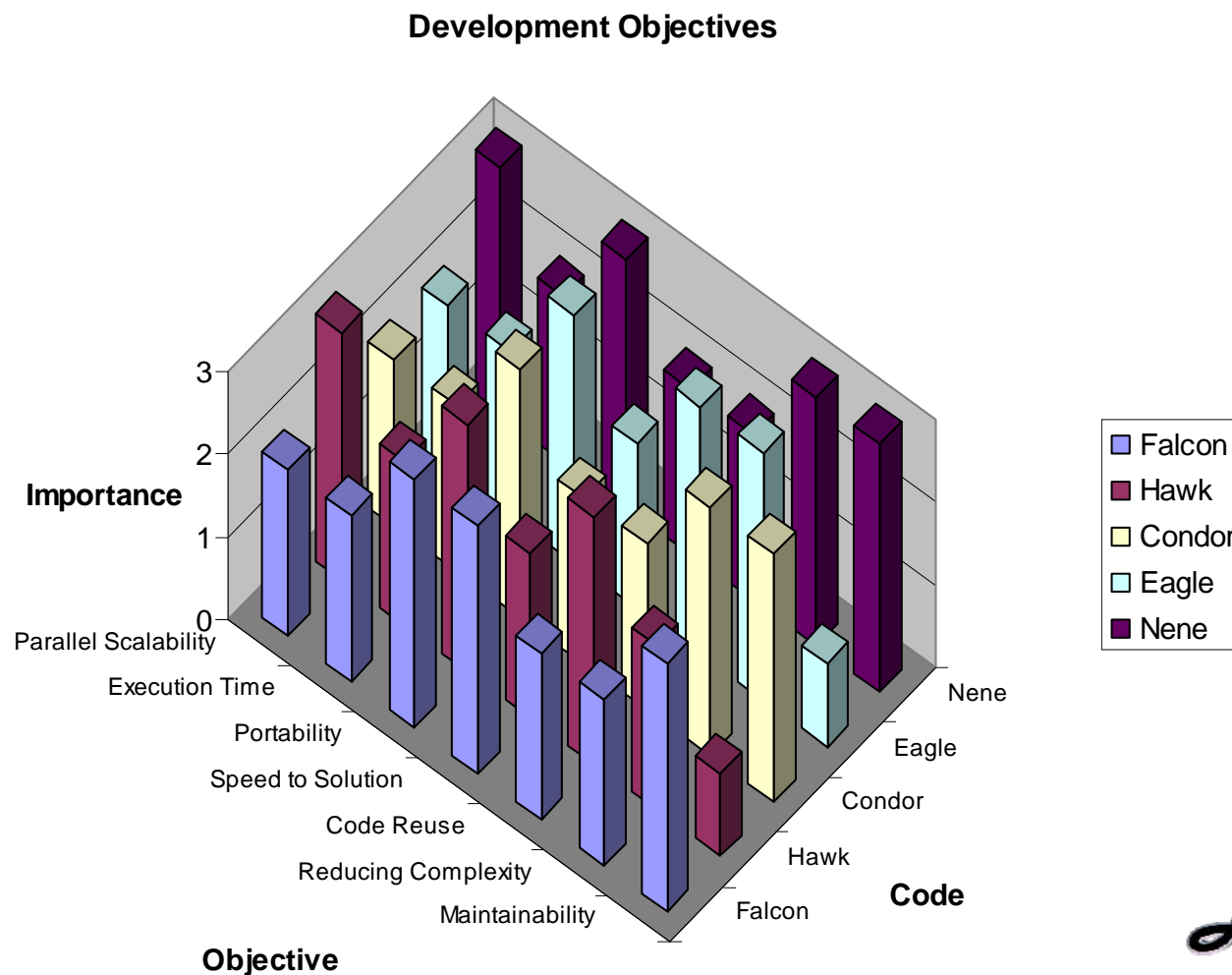
- Third party (externally developed) software and software development tools are viewed as a major risk factor by existing code teams.

The greatest concerns are for parallel debuggers, problem set-up tools, linkers and loaders with ability to link many languages, performance analysis tools, run schedulers, visualization and data analysis tools, testing tools, smoother upgrades to operating systems





Development Objectives





Cross-Study Observations

❖ Observation #5:

- The principal goal of our development teams has been scientific discovery or engineering design.
- “Speed to solution” and “execution time” are not the most highly ranked goals for our existing code teams (except where it impacts the science).

The highest ranked common goals expressed by our case study participants are: codes that work, provide accurate and credible results and are portable.





Cross-Study Observations

❖ Observation #6:

- All but one of the existing codes studied by our team have adopted an “agile” approach to code development without formal software engineering

Hawk, Condor, Eagle and Nene have “agile” teams which emphasized individuals and practices over processes and tools; Falcon was more formal, but no project had formal CMM Level 2 certification





Staffing Profiles

Staff Profile	Falcon	Hawk	Condor	Eagle	Nene
Scientists/Engineers	14	2	3	3	9
Computer Scientists	3	1	0	0	1
Total	17	3	3	3	10





Cross-Study Observations

- ❖ **Observation #7:**
 - For the most part, the developers of existing codes are scientists and engineers, not software engineers or professional programmers.
 - Many developers of scientific codes are also the primary users of those codes.

They tend to be suspicious of rigid software engineering methodologies, preferring the “agile” approach. Even teams with long histories of collaboration do not acknowledge a need to go beyond CMM Level 2—they emphasize good practices rather than good processes



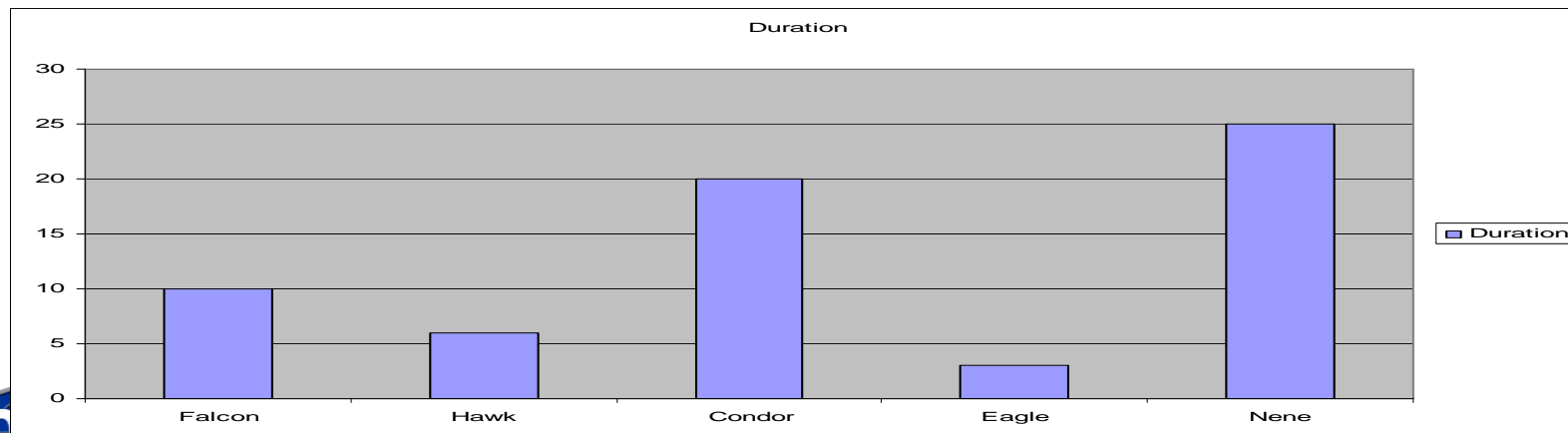


Workflows



General Phases for all Life-Cycles (after Sodhi)

Code	Analysis & Design	Implementation	Testing	Maintenance
Falcon	25%-35%	25%-35%	15%-30%	10%-30%
Hawk	25%	40%	20%	15%
Condor	15%	55%	15%	15%
Eagle	25%	55%	15%	15%
Nene	35%	45%	15%	5%



HPES



Cross-Study Observations

❖ Observation #8:

- Over the lifetimes of the existing codes studied to date, most of the effort has been expended in the implementation workflow step.
- Includes implementation during a long production phase
- A successful computational science and engineering code is undergoing continual development in response to new user requirements





Customer satisfaction, not marketing, determines the success of the code.

❖ Observation # 9:

—The success or failure of a code depends on whether the code team can keep its customers satisfied.

Code teams that helped their customers succeed in their analysis, predictions or research were successful. The code teams that didn't, found that their codes weren't used and were eventually abandoned.





Preliminary Observations from the Nene On-Site Interview

- ❖ Largest project yet (25 years old, ~20,000 downloads, ~100,000 users, 100s of contributors)
 - Over 5100 citations for primary code reference
 - Huge group of satisfied users!!!
 - Best example yet for the dominance of pragmatic *practices* over *processes* in scientific code development
 - Almost no role for formal software engineering
 - Similar to Open Source Model
 - Users download code from website and modify it to solve problems
 - Upgrades negotiated with central team
 - Funding agencies (all the major federal agencies that fund science) provide support for domain science, not explicitly for code development





Development Objectives

	Falcon	Hawk	Condor	Eagle	Nene
Parallel Scalability	Medium	High	Medium	Medium	High
Execution Time	Medium	Medium	Medium	Medium	Medium
Portability	High	High	High	High	High
Speed to Solution	High	Medium	Medium	Medium	Medium
Code Reuse	Medium	High [†]	Medium	High	Medium
Reducing Complexity	Medium	Medium	High	High	High
Maintainability	High	Low [‡]	High	Low	High
[†] For new code [‡] Survey response; the code is highly maintainable, but this was not an explicit design goal					

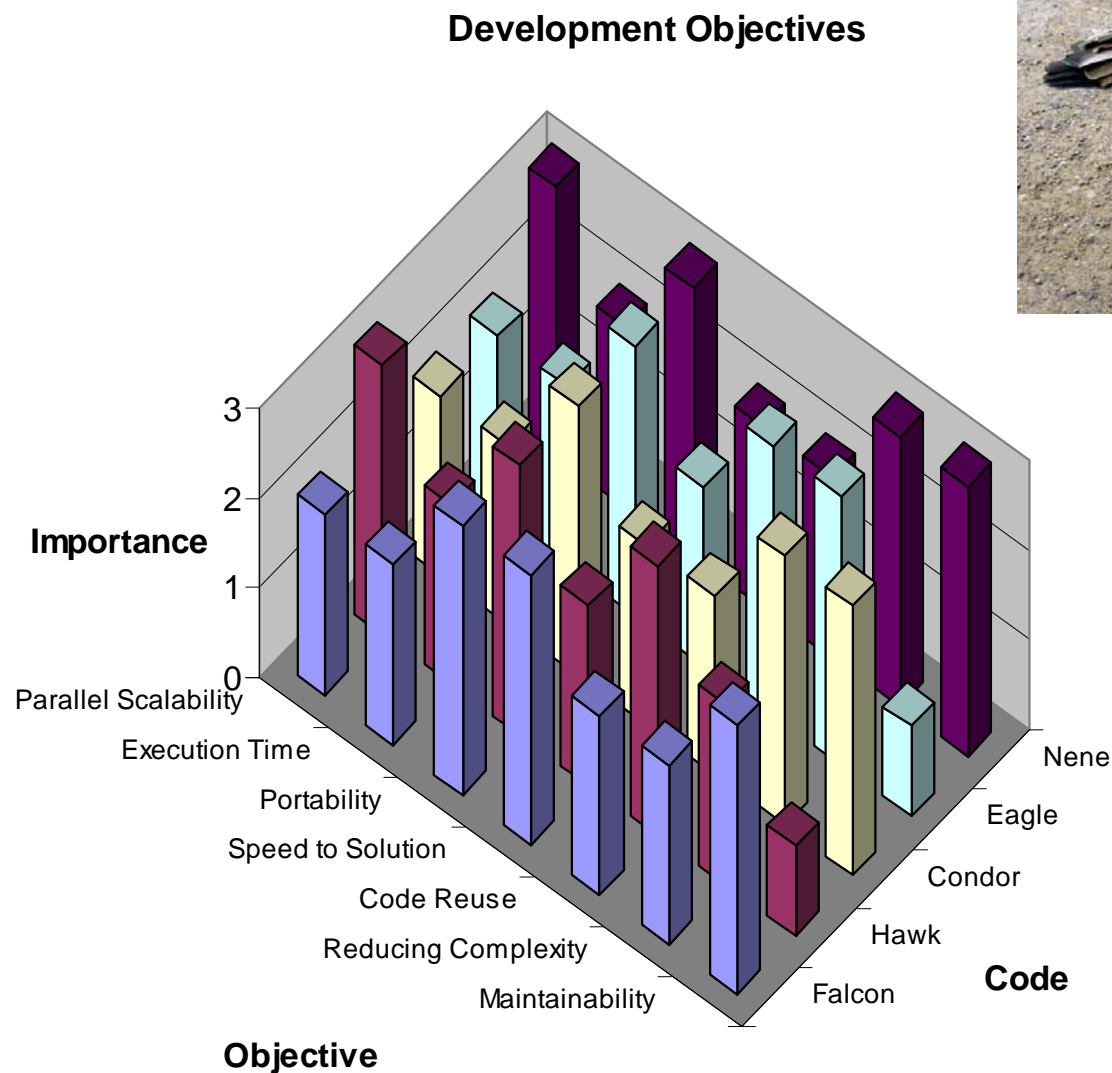


*Medium implies that reuse is occasional
 high implies that reuse is a project imperative





Development Objectives



HPES





Nene Software Development Practices



Practice	Description	Degree Followed
Requirements Development	Produce, analyze and verify customer, project and "product" requirements	Distributed management reduces the need and impact
Requirements Management	Manage requirements and identify inconsistencies with plan	same as above
Project Planning	Establish and maintain a plan that defines project activities	same as above
Project Monitoring & Control	Provide an understanding of the project's progress	No formal plan or deadlines
Configuration Management	Establish and maintain integrity of work products using config. mgt and control	Yes, tight control over program library
Process and Product Quality Assurance	Objectively evaluating adherence to process descriptions and resolving non-compliance	Tight control over contributed capabilities
Organizational Process Definition	Follow an organization-wide process	No, distributed mgt sets its own processes; well-defined process within core team



Practices (Continued)



Practice	Description	Degree Followed
Organizational Training	Develop the skills and knowledge of staff so that they can perform their roles effectively	An important output of this project is the training of graduate students
Risk Management	Identify potential problems before they occur and mitigating them	Long track record of successfully managing risks
Peer Reviews	Software artifacts (requirements, design, code) reviewed by peers to improve quality	Code is reviewed by PI before submission and inclusion into library
Planning Game	Quickly determine the scope of the next release with business priorities and technical estimates	Not relevant
Frequent Deliveries/Small Releases	Frequent releases of the highest priority items	No, delivery occurs when code is ready. Timing is driven by academic calendar



Practices (Continued)

Practice	Description	Degree Followed
Simple Design	Design only what is being developed, little planning for future	Yes, very decentralized planning done by 100's of contributors
Refactoring	Restructuring to remove duplication, improve communication, simplify or add flexibility	Some, especially in areas of active development
Pair Programming	Two programmers work side-by-side at one computer, collaborating on coding	Some, usually only in the feature integration phase, where Pis and students work together and in some cases side-by-side





Practices (Continued)

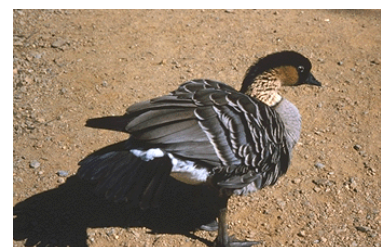


Practice	Description	Degree Followed
Tacit Knowledge	Project knowledge is maintained in participant's heads rather than documents	Yes, a great deal is published, but tacit knowledge is important
Collective Ownership	Anyone can change any code anywhere at any time	No
On-site Customer	Include a real, live user on the development team	Yes, even the core team members are users
Test-Driven Development	Module or method test are written before or during coding	Sometimes



We found sparse use of Software Metrics.

	Falcon	Hawk	Condor	Eagle	Nene
Lines of code	x	x	x	x	x
Function points	x				
Stories, project velocity					
Cyclomatic complexity					
Data coupling					
Comment lines					x
Locality					x
Concurrency					
Defect rates					
Time-to-fix defects		x			
Number of debug runs/unit time					
Test Coverage		x	x		
Frequency that regression testing uncovers problems					
Code performance	x	x	x	x	x
Degree of performance optimization				x	
Parallel scaling		x	x	x	x
Number of users		x	x		x
Number of production runs/unit time					
Computer time for code development/unit time					
Computer time for production/unit time			x		





All of the projects made use of Testing.



Case Study	Falcon	Hawk	Condor	Eagle	Nene
Fraction of Code Tested	~30%*	51-75%	51-75%	>75%	>75%
Conformance between scalar and parallel	n/a	< 2%	no formal bounds		<10 ⁻⁹ Units
Conformance with experimental tests		<32%	no formal bounds		no formal bounds
Verification					
• Compare to exact answer	yes	yes	yes		
• Monitor conserved quantities	yes	yes	yes		
• Preservation of symetries	yes	yes	yes		
• Compare with existing codes	yes	yes	yes	yes	
• Controlled experiments	yes			yes	yes
Regression Tests	yes	no	yes		yes

*Regression Tests



Workflows

General Phases for all Life-Cycles (after

Code	Analysis & Design	Implementation	Testing	Maintenance
Falcon	25%-35%	25%-35%	15%-30%	10%-30%
Hawk	25%	40%	20%	15%
Condor	15%	55%	15%	15%
Eagle	25%	55%	15%	15%
Nene	35%	45%	15%	5%





Reengineering of Legacy Applications

Technology Roadmap Workgroup

Technology Space

1. Program Understanding
2. Modern Replica
3. Testing and Verification
4. Human Interfaces
5. Cost/Productivity Assessment

Program Understanding: Technology Roadmap

- Techniques to identify the abstractions (extraction of abstractions)
 - Identifying known algorithms within applications
 - Identifying data-structures
 - Dependence analysis signatures
 - Static and dynamic analysis
 - Identification of invariants
 - Dynamic dependence analysis
 - Performance optimization
 - Learning (Pattern Recognition)
 - Natural language processing of comments and language construct names
 - Case based reasoning
 - Building the Modern Replica
 - Refactoring
 - Unsound transformations (adapted by programmer or learning algorithm)
 - Program visualization
 - Discovery
 - Data structure based visualization

Modern Replica: Technology RoadMap

- Exciting technologies
 - What can be achieved using them
 - How it can be achieved in 18 months
- However, we found we were
 - “just suggesting we solve many open problems in computer science”

Modern Replica: Goal

- Technologies to remove machine-specific elements of code
 - Certain types of loop unrolling
 - Exploiting pipeline parallelism, for example
 - Data parallelization
 - Pick your best technique
- For replicas, one must:
 - Retain information for optimization
 - Can (re-)generate optimized code
 - Must disallow generated executables
 - Optimization tweaks must be first-class

Replica Construction

- Different languages have different strengths
 - Parallelism, etc...
 - Let the property of interest be your guid
- Model from which to verify properties
 - Determinate behavior
 - Same behavior on exact same input
 - Easy to reproduce defects, a help to debugging
 - Seek provable properties from the replica
- Models of parallelism
 - CSP or DataFlow or ???

Replica

- Challenges in avoid legacy pitfall with the Replica itself
 - Will the Replica be the legacy code in +10yrs?
 - Replica must be easy to analyze
 - General representation of parallelism
- What could have been done differently when legacy code was first written.
 - Early optimization
 - Hopes for automatic parallelism were unfounded. What are we assuming?

Testing and Verification: Problem Statement

- Focused problem: establish that reengineered implementation is equivalent to legacy version
 - Subproblem: establish that modern replica is equivalent to legacy version
- Equivalence w.r.t. behavior on test cases
- “Don’t care” cases e.g., allow modern version to have fewer bugs than legacy version
- Safety net e.g., insert assertions in modern version when assumptions are made w.r.t. error handling

Testing & Verification: Technology Roadmap

- Symbolic execution
 - Test “equivalence” of two versions of same function
 - Overcome path coverage challenge by use of test cases
 - Test for k^{th} degree similarity
- Verification of semantic equivalence
 - Use verifier to check equivalence of constraints from two different executions
- Definition of “don’t care” cases
 - Use test cases to limit behaviors of interest
 - Add assertions to modern replica
- Loops -- overcoming major challenge in verification
 - Use dynamic analysis to distinguish between loops with few vs. large # iterations

Human Interfaces: DARPA-hard problems

1) *Observation*: three kinds of expertise are needed to both develop and port quality HPC code: (i) domain knowledge, (ii) numerical methods, (iii) parallel programming. Either you have one programmer who knows them all or your experts must effectively communicate. [Yes, there is an expertise gap in HPC programming.]

Problem statement: Develop technology and/or social methods that reduce the need for this breadth of expertise or make the communication among experts easier.

Human Interfaces: DARPA-hard problems (contd)

2) *Observation:* Where do specifications/abstractions come from? Programmers are often aware what these abstractions are, but it is tedious or not economical to write them down, because they need to be formally stated. Example: who and how specifies that a library routine sorts the input array?

Problem statement: Develop tools to help programmers infer specifications of modules in their code. Develop incentives that encourage programmers to use these tools.

Human Interfaces: DARPA-hard problems (contd)

3) *Observation:* We believe that porting HPC code is expensive is in part because code transformations are repetitive and performed manually.

Problem statement: Develop methods that automate these transformations. These methods must be easily programmable by the programmer. For example, they can be programmed by demonstration.

Cost/Productivity Assessment: Technology Roadmap

- Use of performance profiles to identify software components that need less vs. more attention from a performance viewpoint
- Ethnographic studies to identify easy vs. hard steps in manual process

Technology Space

1. Program Understanding
2. Modern Replica
3. Testing and Verification
4. Human Interfaces
5. Cost/Productivity Assessment

III Milestones and Evaluation

- **Reduced cost (by 100x to 1000x)**
 - If the process is fully automated → trivial
 - Programmer involvement → user studies?
 - Can we find a set of applications with original legacy version and a version modernized using current practices that can be used as the base case?
- **Reduction of errors and deviations**
 - What is a good measurement?
- **Modernized replica trivially map in to multiple modern architectures**
 - “trivially map”: Automated tools, no programmer intervention
 - “multiple modern architectures”: at least one distributed memory multicore (cell or Tile64) and one shared memory multicore (core 2 duo or niagara)
- **Modernized replica is efficient and effective**
 - Show speedups against the original program on that architecture
 - Show scalability from one core to max number of cores available
- **Modernize replica can be easy to understood and managed**
 - How to measure quality of the specifications created?
 - How to measure malleability and extendibility of original vs. modernized replica?
- **Room D451**

Metrics

- **Performance**
 - Speedup on several architectures
 - Minimal performance level for acceptance; not used for comparing teams
- **Productivity**
 - Measured in programmer time, not SLOC produced
 - Primary metric for comparison of teams
 - Depends on expertise of programmers
- **Maintainability**
 - Code size of modernized replica
- **Flexibility**
 - Replica easily targeted to new architectures with new capabilities

Phase 1 Benefits Measurement

- **A referee team establishes the rules of the game**
 - Example:
 - a benchmark program (possibly several, since proposers may use different source languages); ~10k SLOC?
 - multiple target architectures
 - scalability improvement: 2x faster than unchanged legacy code on target arch
 - maintainability improvement: 2x smaller SLOC than legacy code
 - proposers have a week to train a small team of programmers (grad students?, independently hired) on their tool
 - programmers have a month to do the port to the modern replica
 - measure time to do the port
 - Metrics for other DoD projects are available
 - Reuse their procedures?
- **Overall goal: 2x reduction in programmer time relative to best-practices hand porting**

Phases and Milestones

- **Phase 1 (first 18 months)**
 - Deliverable: prototype process and tools
 - Measurement: 1-month comparison trial
 - Goal: 2x improvement in speedup, porting time, code size
 - Critical design review
 - Cut down proposing teams
- **Phase 2 (next 18 months)**
 - Goal: 10x aggregate improvement (scalability, portability, size)
 - Flexibility: how much time to take advantage of a completely new architecture?
 - Larger codes: 6-month comparison trial using 50k SLOC?
 - Cutting down teams to one representation (selected for handoff to a standardization process?)

- **Proposer picks:**
 - the modern replica representation
 - it should probably already exist, because Phase 1 isn't long enough to develop it
- **Referee picks:**